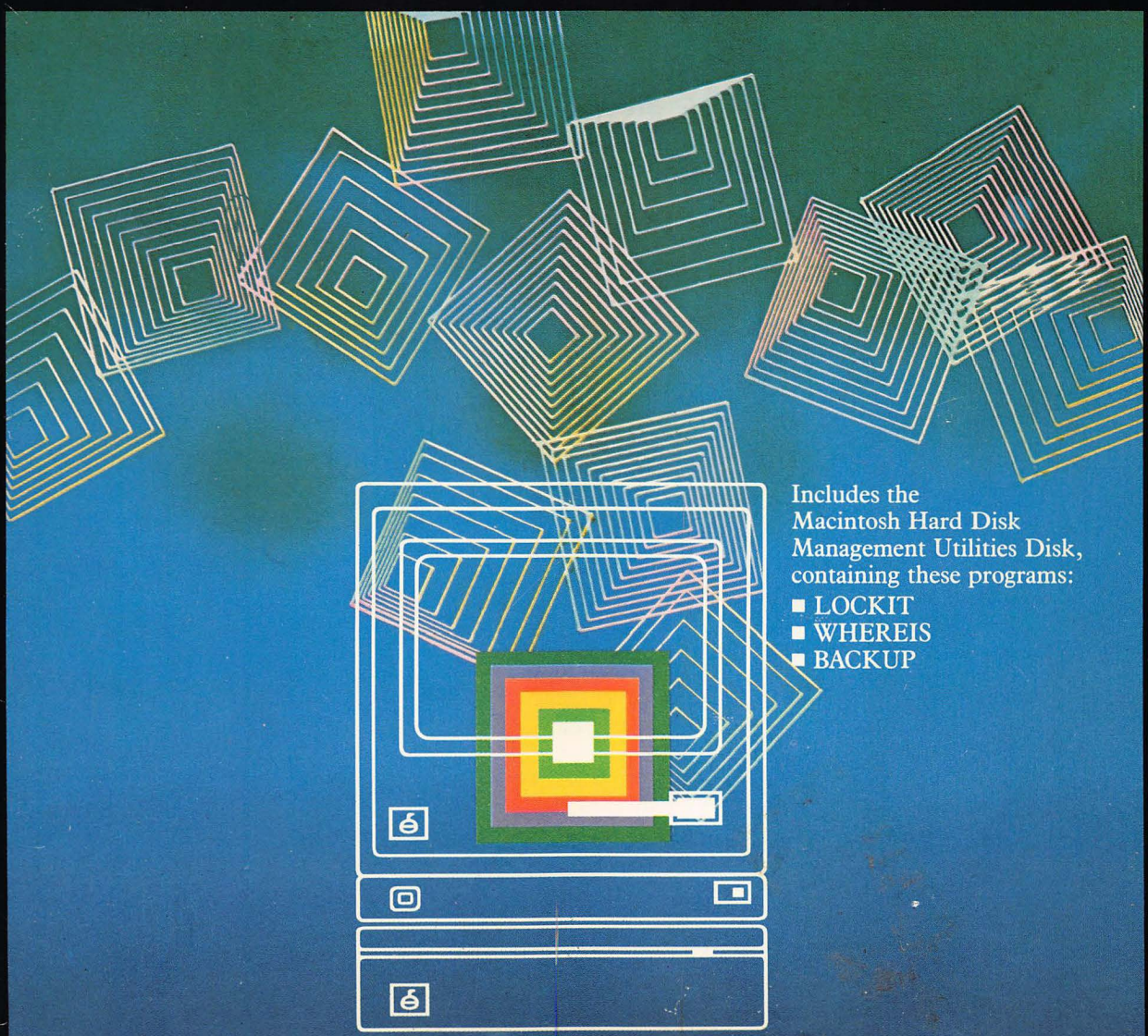
BANTAM COMPUTER BOOKS

THE MACINTOSH PERFORMANCE LIBRARY

DISK INCLUDED

HARD DISK MANAGEMENT FOR THE MACINTOSH

NANCY ANDREWS



Includes the Macintosh Hard Disk Management Utilities Disk, containing these programs:

- LOCKIT
- WHEREIS
- BACKUP

Hard Disk Management For The Macintosh

Hard Disk Management For The Macintosh

Nancy Andrews

BANTAM BOOKS
TORONTO • NEW YORK • LONDON • SYDNEY • AUCKLAND

Hard Disk Management For The Macintosh
A Bantam Book / June 1987

Interior design by Mike Kelly and Margaret Fletcher, Slawson Communications, Inc., San Diego, CA.
Production by Slawson Communications, Inc., San Diego, CA.
Illustrations by Nancy Andrews and John McAusland

ISBN 0-553-34398-X

Published simultaneously in the United States and Canada

Bantam Books are published by Bantam Books, Inc. Its trademark, consisting of the words "Bantam Books" and the portrayal of a rooster, is registered in U.S. Patent and Trademark Office and in other countries. Marca Registrada. Bantam Books, Inc., 666 Fifth Avenue, New York, New York 10103.

PRINTED IN THE UNITED STATES OF AMERICA

0 9 8 7 6 5 4 3 1

Acknowledgments

Special thanks to the talented group at Bantam Books. Specifically to Jono Hardjowirogo for getting the project going, for his tireless efforts to get the necessary software and hardware, his encouragement, good editing, and sense of humor. Thanks also to Publisher Kenzi Sugihara for his behind-the-scenes support and to Bruce Sherwin for managing this book through production.

Thanks, too, to those who helped me on this end — to my friend and teacher, Jim Tomlin, to Martha Steffen, Scott Schnell, and the technical group at Apple, to Bill Gladstone, and to Ron Aldrich.

And thanks to my family for once again putting up with, and in fact supporting, my single-minded devotion to this project.

Contents

Introduction

Cocktail party questions have changed. People used to ask what kind of computer to buy. (I recommend the Mac with a hard disk.) Now they're asking questions about their hard disks. At a party just last week a friend enlisted my help, admitting her hard disk had gotten so out of hand she only used it for applications and stored all of her documents on floppies so she'd know where they were. This book is for her and others like her. It explains how to manage a hard disk. What this means is making it work for you rather than being at its mercy.

A hard disk allows you to store massive amounts of information and retrieve it in seconds. Managing a hard disk means planning, organizing, and controlling its use. A well-organized hard disk can save you time and money, enabling you to provide higher quality services. A poorly organized hard disk can cost you time spent searching through hundreds of files for a poorly labeled document or one you neglected to put in the proper folder and obviates the increase in speed and

convenience a hard disk offers. Failing to back up the information on your hard disk can mean the potential loss of hundreds of files as well as thousands of hours of recovery time.

A GAME PLAN

This book starts with the basics. Chapter 1 introduces you to hard disk terminology and technology and gives you some practical information on the care and feeding of your hard disk. Chapter 2 compares different types of hard disks — internal and external, ones that connect to the serial port and ones that use the SCSI port. It's useful if you've not yet purchased a hard disk or if you're ready to invest in a second one. Chapter 3 explains hierarchical directories and the Mac's hierarchical file system (HFS). This is the key to organizing your hard disk. If you're new to the Mac and hard disks, you'll need the information in the opening chapters; if you're a seasoned Mac user and have experience with hard disks, just skim these chapters.

The next three chapters take you through the nuts and bolts of organizing your applications and documents on your disk and then using it. Chapter 4 discusses basic disk organization, then presents several specific organization plans, and shows how to implement them. Chapter 5 presents guidelines for basic hard disk use and chapter 6, tips for optimal use.

Chapters 7, 8, and 9 give you three utilities you can use to manage your hard disk. Chapter 7 presents a program called LOCKIT, a password protection program. Anyone who wants to use information on your hard disk must first supply the correct password. Chapter 8 presents WHEREIS, a program to locate missing files. Chapter 9 shows you how to construct a BACKUP program, a utility that looks at file dates and only backs up files that have changed since the last time you backed up. These chapters provide Pascal program listings for

these utilities. If you have a Pascal compiler and want to learn a bit about Mac programming, you can look at the source code customize and enhance it for your need. You can use the utilities on the disk just as they are.

Chapter 10 introduces you to networking and sharing disk space and printers. It takes a brief look at three different kinds of networks — MacServe, HyperNet, and TOPS. Chapter 11 takes an in-depth look at one specific network, AppleShare, Apple's network file server and workstation software.

The last chapter contains information you hope to never have to use. Chapter 12, What If . . . , contains trouble-shooting and maintenance instructions. It covers questions like what is a hard disk "crash"? can you recover? how much? and how soon?

When you finish the book, have set up and organized your hard disk to work the way you work, and are beginning to enjoy its speed and convenience, you won't think about your hard disk much any more — you'll just use it.

CHAPTER 1

Disks, Drives, and Disk-Related Issues

A hard disk can store all the details your Mac must remember in one place — from your 1987 second quarter budget to your assistant's Social Security withholding tax in 1959. In the paper world its equivalents are telephone directories, filing cabinets, reference books, and ledgers. But a hard disk is more than just a replacement for 25 to 50 floppies. The major advantages of adding a hard disk are speed and convenience.

ADVANTAGES

Speed

Floppy disk drives are slow. Memory locations are read by a head attached to a mechanical arm that sweeps across the disk. It takes

time for the head to start, stop, and settle down to read. Delays seem interminable and stretch out even simple jobs.

A hard disk spins about ten times faster than a floppy disk. In addition to this, a hard disk is constantly spinning (that's the noise you hear) while a floppy disk starts and stops each time it is used. This saves the time it takes for a floppy disk to get up to speed when the drive starts. Any time you hear the floppy drive whir, a hard disk will be faster.

Although it's interesting to know how much faster a hard disk spins, what you're most likely more interested in is where you'll actually see speed improvements as you work. Most comparative studies of different hard disks show the time it takes to start and quit large applications. But if you work with a small number of applications, a few seconds saved at the beginning and end of your work is not particularly significant. What is more important is the time you save while working on an application.

Large applications store only a part of their code in memory. Then when they need another piece, they read it in from the disk. While the application is reading from the disk, other processing stops. Since the disk access is so much faster with a hard disk, delays in program functioning are practically unnoticeable.

Most applications store only a part of a very large document in memory at one time. For example, if you're working on the end of a very long report and you need to scroll to the beginning to check your opening sentence, the application needs to read that part, in this case the beginning of the document, from the disk into memory. Using a floppy system, you'll notice the program pausing to get this information, while with a hard drive you'll barely notice a delay.

Another place you'll certainly notice speed improvement is in opening and saving documents. This is significant if you work with the same application but with several documents. For example, if you're a

writer and use a word processor to work on several pieces a day, you won't have time to think about whether you should get up and pace while waiting for the Mac to close one document and open the next.

Here's one last obvious time-saver. Because most likely all your data and applications will be on the hard disk, moving the disk's head to the location on the disk for the file you need is definitely faster than the human arm removing and inserting a floppy disk.

Convenience

People who really "need" a hard disk are people who work with large databases and spread sheets. If files are larger than will fit on a floppy, they need the storage and speed of a hard drive. But if you're neither a pack rat nor a speed demon, convenience alone may justify a hard drive.

A 400K floppy will hold approximately 200 pages of text, an 800K floppy 400 pages, while a 20 MB hard disk will hold 10,000 pages! With a small inventory database of 200 items and 200 transactions a month, you could store six months of inventory information on a 400K floppy and a year's worth on an 800K floppy. On a 20 MB hard disk you could store inventory information for 25 years. (That is, of course, if you have nothing but inventory records on the disk.) The convenient part of all this storage space is that you can totally eliminate the irritating swapping messages, time-consuming disk-swapping, and interminable, tedious shifting of information from one disk to another.

In addition, you'll always have the desk accessories and fonts accessible. You'll no longer pull down the Apple menu only to discover that the desk accessory you want is not there and is most likely on a different, perhaps forgotten, floppy. You're allowed up to 16 desk accessories and there's plenty of room on a 20 MB disk to hold the full 16. In

addition, you can store all of your fonts and have them always available (up to 65,536 of them). The font menu with System version 3.0 or higher scrolls displaying your entire collection. These, like your desk accessories, will always be accessible, regardless of which application you're working with.

If you've worked with Switcher without a hard disk you know that theoretically you can have four active applications if you have a fat Mac and eight with a Plus. But practically you can never actually run that many applications because you run out of room. In addition to the applications you want to switch between, you also need room for the Switcher program and the System and Finder. Most of the time you run out of disk space before everything is set up perfectly. With a hard disk that has all your applications, Switcher, the System, and Finder in one place, you can set up Switcher exactly the way you want it and you won't run out of room.

There is no question that a hard disk costs more than the 25 to 50 floppies it replaces. But if you regularly use several programs and work with many documents, you'll find a hard disk nearly essential. And the convenience of having Finder search through a myriad of files and locate the one you need, as well as the increase in speed, justify the cost.

After extolling the convenience of using a hard disk, it's only fair to mention two inconveniences — copy protected software and noise. Although you can copy all applications onto your hard disk and run them from the hard disk, some require you to insert the floppy master disk when you start the application. This is annoying and still requires you to keep a small stack of floppies beside your Mac and search through them for the appropriate disk. Fortunately the trend is moving away from copy protected software. Both Microsoft and Ashton Tate have removed their copy protection; hopefully the other players will follow suit. (Chapter 5 discusses temporary solutions for running copy protected software from a hard disk.) All hard drives add fans for

cooling. The fans and the constant spinning of the disk add noise. Some are considerably noisier than others. It's not a bad idea to listen to the one you're considering before purchasing.

CARE AND FEEDING . . .

If you know a little bit about hard disk technology, caring for your hard disk properly will make sense.

All disks have a magnetic recording surface. Data is stored on the disk as magnetic fields created by the movement of the read-write heads across the surface. When floppy disks operate, the read–write head skims the disk. Friction causes the disk to wear out and lose its ability to record magnetic fields without error. That's why floppy disks periodically fail and it's important to make frequent copies.

Hard disks have smaller rotating heads. This allows greater precision of movement so data can be recorded more densely. With a hard disk wear is not a problem because the disk head rides a hair's breadth above the disk surface — 12 to 18 thousandths of an inch (see Figure 1-1).

However a stray piece of dust or a smoke particle can bump the head, causing the head to touch the disk, and ruin a part of the disk. This is the ominous "head crash". Fortunately hard drives are tightly sealed in airtight chambers to keep contamination out.

Because the head touching the surface of the disk can cause problems, it is important to move the hard disk carefully and to position it where it will not be bumped or jostled. Early hard disks had a utility you could use to park the heads very close to the center of the disk if you were going to move or ship the disk. That way if the heads touched the disk surface, they would touch in a place where no data

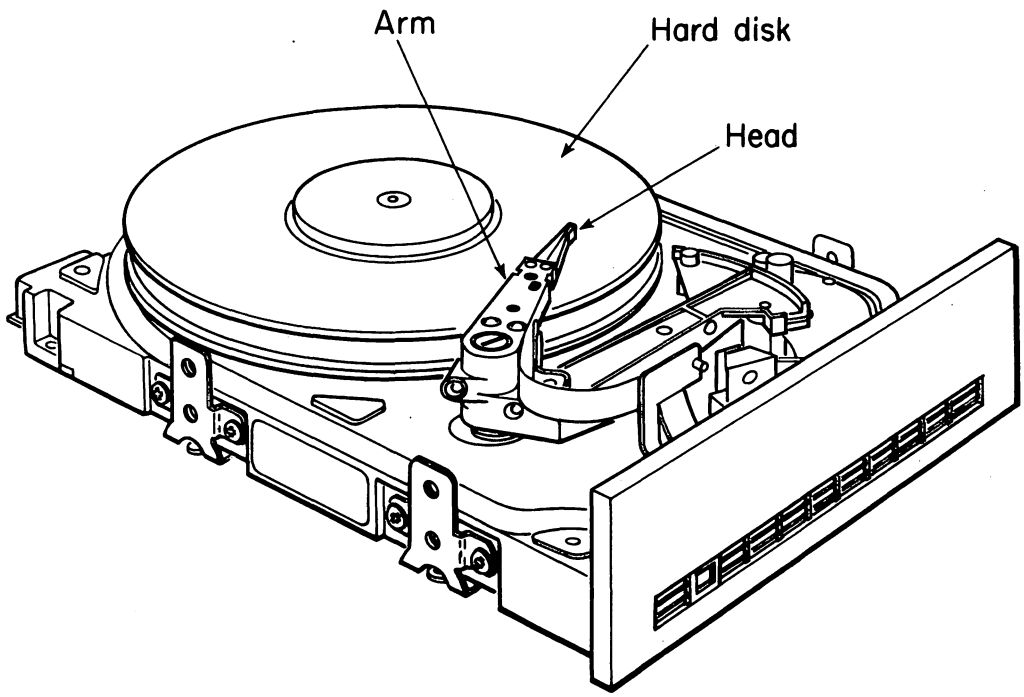


Fig. 1-1. Inside a hard disk drive.

was recorded. Since then the technology has improved; the arm is more stable, and the utility is no longer required. Nevertheless, if you must ship your disk, if you have an external disk, be sure to package it at least as well as it was packed when you got it. If you have an internal disk, a heavy-duty case is the safest. Some internal disks packed only in the Mac's original boxes were damaged when shipped.

Apple provides very little information about caring for its hard disk (HD 20): only the obvious — don't spill liquids inside or expose it to chemical or solvent fumes. Its definition of chemical or solvent fumes

includes steam from a coffee maker! Basically you should treat your hard disk gently, just as you do your computer or any other piece of valuable equipment. If your computer typically gets rough treatment, it may be a good idea to stick to floppies.

DIFFERENCES

All hard disks have similar convenience and speed advantages and require similar care, but this is where the similarities end. There is an additional Mac in the family, the Mac Plus, many different hard drives both internal and external, drives that connect to the external drive port, the serial port, and ones that use the SCSI port, and a new System and Finder — ones that supports hierarchical files. All of these changes have implications for the type of hard disk you choose or for managing the one you own. Information about specific drives follows in chapter 2, but the next section introduces the basic issues.

The Mac Plus

The Mac Plus was optimized for the addition of a hard disk. It has new ROM twice as large as the old ROM — 128K instead of 64K. What this means is that although the Plus functions much like the 128 Mac, because some of the routines previously stored in the System are now in ROM, the Plus is faster. This is because ROM, unlike disks, has no moveable parts — it is electronic rather than mechanical — so reading from the ROM is quicker than getting the same information from disk. In addition the new ROM has a driver for the HD 20 and allocates space for a disk cache — an area of memory used for storing disk information. What the disk cache does is store information from the disk that the application uses repeatedly; this makes fewer disk accesses necessary and speeds up the application. (Instructions for using the disk cache are in chapter 7.) The 128K ROM also includes code to let the Mac start directly from the hard disk; prior to this you

needed to use a startup floppy for a hard disk connected to the serial port.

In addition to the larger ROM, the Plus also has an 800K floppy drive, a Small Computer System Interface (SCSI), a new keyboard that includes a numeric keypad and cursor keys, and 1 MB of RAM you can expand to 4 MB's. Fortunately owners of 128 and 512K Macs aren't stuck with obsolete equipment. Apple made it possible to upgrade old Macs to the Mac Plus by purchasing the new ROM and disk drive and a new digital board with the additional memory and the SCSI port.

Internal and External Drives

Even though several other companies had introduced hard disk drives for the Mac earlier, when General Computer introduced the first internal hard disk for the Mac, the HyperDrive 10, it became popular. Your dealer had to install it. It fit inside the Mac and clipped on to its 68000 processor. There were no choices — if you wanted an internal hard disk drive, you got a HyperDrive.

Then more manufacturers followed suit. Some were internal drives like HyperDrive and attached to the Mac's microprocessor; others were external like the Apple HD 20 and were connected via cable to the Mac's serial port or external drive port.

Initially, if you wanted a fast hard drive, you got an internal drive. External drives connected to the serial port just couldn't compete with the internal drives connected directly to the microprocessor. But internal drives were complex and fragile and packed into a computer not designed for this type of modification. And some internal disk installations voided the Mac's warranty.

SCSI or Serial

The Mac Plus and Mac Plus upgrade added a Small Computer System Interface (SCSI — pronounced "scuzzy") port. This port is an industry standard port used to connect high speed peripherals such as hard disks, scanners, and tape backup systems to the Mac. You can use this port to daisy chain up to seven peripherals to your Mac (see Figure 1-2).

There are two advantages to the SCSI port. First it provides a much faster way to get information from the hard disk to the computer. Hard drives that connect to the SCSI port are typically much faster than those that connect to the serial or external disk drive port.

Some studies indicate they are as much as six times faster. And there are a whole generation of hard drives that take advantage of the SCSI port (Chapter 2 has more specific information about SCSI drives). The second advantage is that it makes the Mac expandable. Theoretically you can add any peripheral that uses the SCSI interface. But because each manufacturer implements his or her own version of this, in practice what you need to do is add a SCSI cable that fits the Mac's 25 pin port and a software device driver for the equipment you're adding.

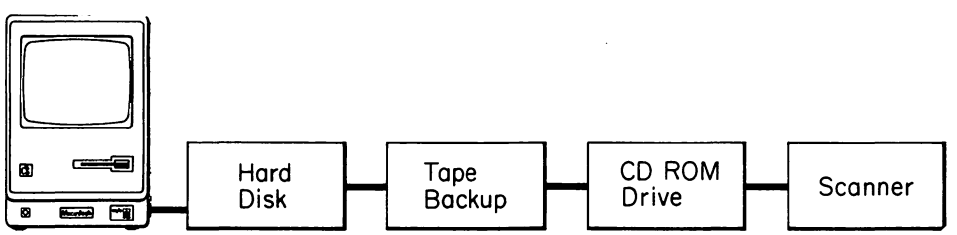


Figure 1-2. Daisy chaining peripherals to Mac via the SCSI.

MFS and HFS

When the Mac first arrived it used MFS, the Macintosh Filing System. It was a system designed to work with the quantities of files found on floppy disks. It was called a flat file system, because although you could put files away into folders, this was essentially just cosmetic — the System stored all files at the same level.

Because a 20 MB hard disk could contain 100 times more files than a 400K floppy disk, the original Macintosh Filing System was not practical with a hard disk. Scrolling through hundreds of file names after choosing an application's Open command is incredibly slow and inefficient.

So Apple implemented a Hierarchical Filing System, the HFS, a much more effective system for managing large numbers of files. What it does is organize files — applications and documents — into folders, but these folders can be many levels deep. It is analogous to an office filing system that uses hanging files. One folder would be labeled Personnel. Inside the Personnel folder would be a folder for each employee. Inside each employee's folder would be an employment application and payroll, performance, and emergency information documents. To find an emergency contact for employee Horatio Toad, you'd first open the Personnel folder, then employee Toad's folder, and inside that open the emergency information document and locate the emergency contact. You'll find that with a little practice it's easy to use (see Figure 1-3).

In addition to letting you nest folders, the HFS knows about folders. The standard file dialog box, the one you see when you open and save documents, only lists the files and folders in the current folder. What this means is that even if you have 10,000 files on your hard disk, you don't have to rummage through a list of all of them to find what you want. Finding a file on a crowded disk is easier and the Finder and disk drives have less work to do — instead of displaying an entire list of all the files on your 20 MB disk, they can instead concentrate on the current folder.

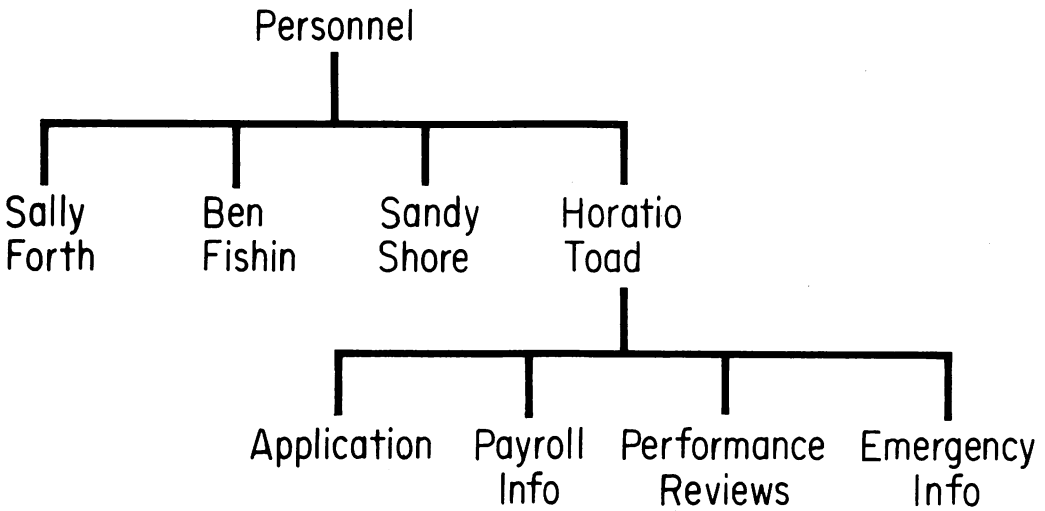


Figure 1-3. The HFS documents and folders nest inside one another.

The HFS also provides a way to move from folder to folder while still in the standard file dialog box. If what you want is not in the current folder, you don't need to cancel the command, open folder after folder until you find what you want, and start again. Instead you can move from folder to folder and look inside different folders all within the standard file dialog box. An additional benefit of hierarchical files is that files need not have unique names. Two or more files or folders can have the same name as long as they reside in different folders. This would make it possible, for example, to use the folder names Quarter 1, Quarter 2, Quarter 3, and Quarter 4 as nested folders inside an income folder and folders with the same names inside an expense folder.

Chapter 3 contains a complete description of the HFS and how to use it, as well as strategies for organizing folders and files on a hard disk.

The HFS was introduced in January 1986 before the Mac Plus with Apple's HD 20 (as System 3.1 and Finder 5.1). At that time it was in software. With the Mac Plus or ROM upgrade, code for the HFS has been put into ROM and works much more quickly. The HFS works well without the new ROM (on a fat Mac with a hard disk, for example), but it will be slow.

Changing from the MFS to the HFS has implications you may not be aware of. Although HFS is a much-needed way of organizing space on a hard disk and Apple designed it to be upwardly compatible with the MFS, some software, particularly if it was written before January 1986, may not work well with HFS. Most major applications run well, but some programs require all application files and documents in the root directory. If you've not yet used HFS, check to be sure the software you use regularly supports the HFS. If not consider a hard disk that can be partitioned. That way you can make an HFS partition for most of your work and an MFS partition for any application that won't work with HFS. (Chapter 2 has information on specific hard disks.)

There is no question that the addition of the Mac Plus with the HFS, the new ROM, and SCSI ports has increased the choices surrounding hard disks. But although choosing a drive is more complex, the drives now available are faster, more efficient, and more economical. The SCSI interface makes them faster and priced competitively with IBM drives, and the HFS incorporated into the new ROM makes them more efficient and easy to use.

Although SCSI drives are currently the latest and fastest, optical drives may follow soon. Optical drives use lasers to read and write data and can store much more data than traditional magnetic disks. Currently optical disks for PC's store 100 to 200 MB's. But optical disks aren't erasable. Information stored on them is permanent. Another name for them is WORM drives — Write Once, Read Many. Because information on these disks cannot be changed, they are unsuitable for many of the applications that use magnetic disks and will most likely not replace them.

CHAPTER 2

Hard Disk Comparisons

To prepare to write this chapter I used four different types of hard disks. I own a Macintosh HD 20. General Computer lent me a HyperDrive 20 (and a Mac to go with it), Peripheral Land lent me a MacFast 20, and Adon Corporation, OEM manufacturer for MDIdeas, lent me a 30 megabyte MDIdeas hard disk with an Adon tape backup system. I used each of these disks over a period of two months so I would know first-hand how they work.

Each is a different type of disk. The Macintosh HD 20 connects to the external drive port which is essentially a serial port. This lets it work with any Macintosh — 128K, 512K, or a Mac Plus. HyperDrive 20 is an internal hard disk. It must be installed inside the Mac by a dealer. It, too, will work with any Mac. Peripheral Land's MacFast 20 is a SCSI drive. This means it attaches to the SCSI port of the Mac Plus. It can

only run on a Mac Plus or an upgraded 512K Mac. The MDIdeas drive is also a SCSI drive and needs a Mac Plus or an upgraded 512K Mac. It has a tape cartridge system built in which provides an easy, painless solution to the backup problem.

I used each of these disks and attempted to evaluate them on these criteria: speed, noise, flexibility, reliability, available utilities, and support. My speed comments are largely intuitive. Milliseconds of data transfer are really not meaningful because the data transfer rate is just a small part of the total transfer time. And because I had different hardware configurations — a Mac Plus for the Hard Disk 20 and the two SCSI drives, but a 512K Mac (not upgraded) for HyperDrive, it didn't seem fair to run a timing test. Instead I used each disk and compared it to other disks I was using for normal every day work.

MACINTOSH HARD DISK 20

The Hard Disk 20 fits conveniently under the Mac and is simple and straightforward to install and use. It attaches to the Mac's external drive port rather than to the SCSI port.

Speed

The Macintosh HD 20 was the slowest of disks I used, but was still acceptably fast. It was two to four times faster than floppies, I didn't have to worry about finding, inserting and swapping floppies, and I wasn't bothered by the speed (at least not until I tried the much faster SCSI drives).

Noise

I found the HD 20 acceptably quiet. It does have a fan that makes noise, but it blends into the background. My partner, on the other

hand, found the fan irritating. He chose to work from floppies rather than to listen to the fan. My advice is to listen before you buy. If the HD 20 suits you except for the noise, you can get a joystick extension cable from Radio Shack and place the disk farther away.

Flexibility

The HD 20 can only be an HFS disk. You can't partition it and designate an MFS partition and an HFS partition. I haven't found this to be a problem, but I don't run any appliations that won't run under HFS. If you do, you may want to choose a more flexible disk.

You can connect one additional HD 20 to the back of the first one, giving you a total of 40 megabytes.

Reliability

I found the HD 20 incredibly reliable. It traveled hundreds of miles in my car trunk. I used it at the ocean, at home, in my office, in my programmer's office, and in a contractor's office. I carried it around instead of floppies and it never once failed.

Utilities

The HD 20's utilities are easy to evaluate — there are none other than the basic format and disk test program. The disk test program tells you if you have a serious problem, if you need to reformat, or if you need to "contact your dealer for repairs". But the HD 20 has no backup/restore utility, no file manager utilities, and no password protection.

Support

As I mentioned above, I never needed support. It was incredibly reliable. And because this is an Apple product, one can assume support

is the same as with other Apple products — that is, it is essentially as good as your dealer.

Overall

I think of the HD 20 as the VW beetle. It gets you where you need to go, maybe not in style and with top speed, but does do the job reliably and consistently.

HYPERDRIVE 20

The HyperDrive 20 fits inside a Mac as seen in Figure 2-1. Mine was installed inside a 512K Mac, not a Mac Plus. Compared to the Mac Plus the 512K Mac is not very fast, and without the upgrade, HyperDrive only runs MFS. What this means is that you cannot nest folders. You create partitions or drawers that are like individual disks. Then you mount the drawers you need. Desktop looks like Figure 2-2.

If you start an application in one drawer and want to open a document located in another drawer, you click the drive button and work your way through the mounted drawers until you come to the one with the document you need. If it turns out the document is in one of the drawers you haven't yet mounted, HyperDrive has a Drawers desk accessory you can use to mount any drawer.

Speed

HyperDrive 20 is actually very fast. It fits right on the 68000 processor and has good startup code so it starts quickly. Because of the time it takes to rebuild the desktop when you quit an application, HyperDrive

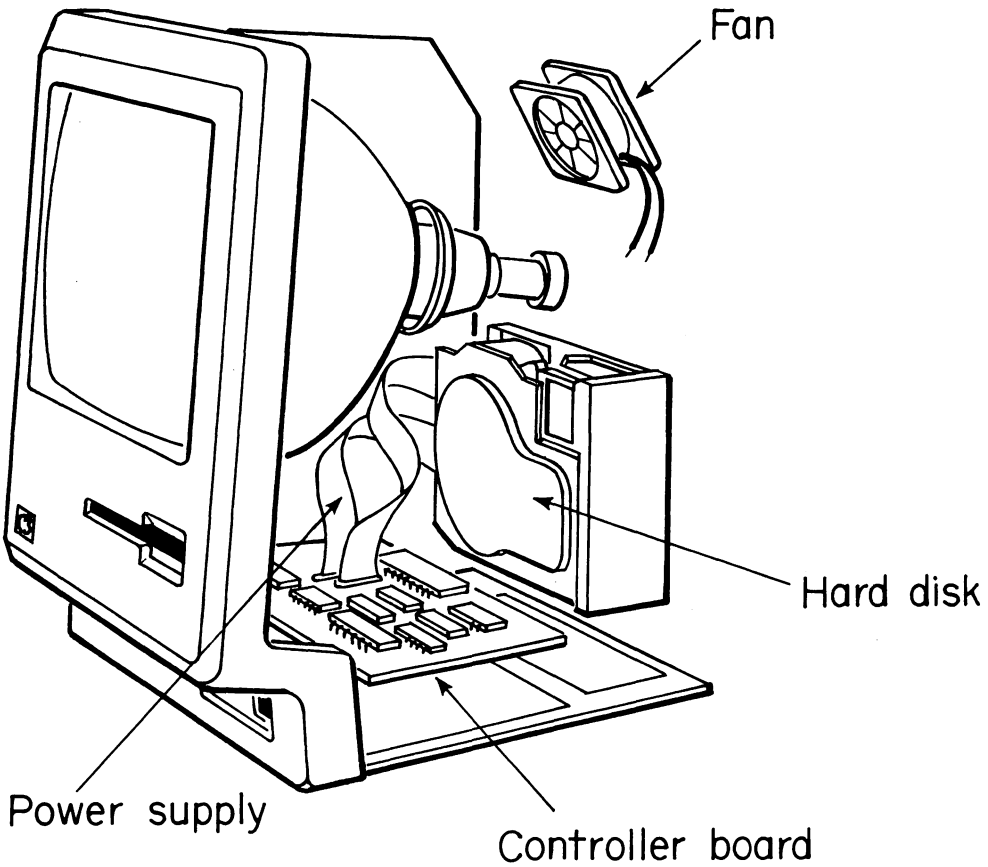


Figure 2-1. Internally mounted hard disk drive.

works faster with fewer drawers mounted. A good rule of thumb is to only mount what you "always need" at startup. Mount others as you need them and put them away when you're finished. HyperDrives, like mothers, reward keepers of neat desktops.

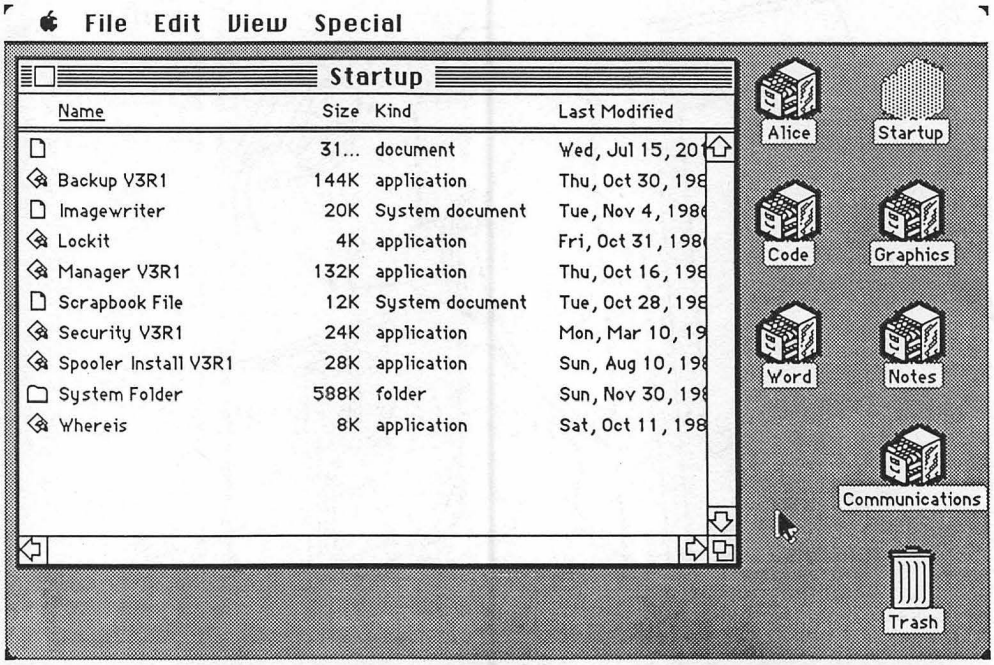


Figure 2-2. HyperDrive desktop.

Noise

HyperDrive 20 is the quietest of the drives I used. Because it fits inside the Mac, even though it has its own fan, it is very quiet. You most likely won't even know it's running.

Flexibility

HyperDrive 20 partitions the disk space into drawers. It allocates space to the drawers in 256K chunks, and it dynamically expands drawers for you if you need more space. This is handy. You don't

have to determine a fixed size in advance or go in and change the size of a drawer after a "disk full" message.

With a HyperDrive in a 512K Mac I could only run MFS. HyperDrive's drawers are a nice solution to the limitations of MFS, but HFS is an even better solution. And some applications such as Apple's Macintosh Programmer's Workshop (MPW) would not run without HFS.

Because of the way HyperDrive is installed, you can only have one HyperDrive per Mac.

Reliability

The HyperDrive I had worked well for about three weeks. Then it seemed to disintegrate. It would take four to five minutes to start up, two + minutes with the happy face Mac and two + minutes with the HyperDrive logo. Typically these are just on the screen for seconds. But once the drive was started, it worked well. I thought perhaps the boot sector part of the drive was failing, so I backed up everything on the disk, reformatted it, and then restored my files. This fixed the problem for another three weeks. Then it returned. In their defense I must say that this review Mac and Hyperdrive may have been shipped to and used by many reviewers, and shipping is not kind to hard drives. So my experience may not be a good test of HyperDrive's reliability.

Utilities

HyperDrive gets the prize for the best set of utilities. There's really no contest. First of all they have a utility to password protect each drawer. That way if several people share the Mac you can keep people from snooping around your drawers. There is a Manager utility to create and delete drawers, to optimize the disk, and to test and initialize it. Also included is a print spooler. It allocates a part of the disk to store

documents you want printed and then prints them in the background, freeing your Mac for other tasks.

Finally, HyperDrive has an elegant backup utility. You can use it to backup the entire disk, selected drawers, single files, or only files created or changed since the last time you did a backup. It backs up to floppies, displays icons for the number of floppies you'll need, and then shows each floppy icon filling up as the backup operation progresses. (It's cute.) It has as its corollary a restore program for returning the files to the hard disk. Again you can return everything to the hard disk or restore only selected files or drawers. When HyperDrive does a backup, it creates a master file. It uses information in this file to restore from the floppies. Although this usually works well, a disadvantage is that if anything happens to this master file, information on the floppies cannot be used without the master because it is not stored in a format the Finder can use. The backup utility also prints reports detailing which files were last backed up and restored. It can even compare a file on the HyperDrive to one on the backup floppy and report any differences.

Support

General Computer does have a technical support line to answer questions. I called them several times. When they didn't have an immediate answer, they did call back with the information I needed.

Overall

The HyperDrive is fast and quiet. It's a good choice if you need to conserve space or if you have a Mac without a SCSI port. It's disadvantages are that it can't be moved and used with another Mac, and if it needs repairs, you must send your Mac, too, Also, its reliability is somewhat questionable.

MACFAST 20

The Peripheral Land MacFast drive comes in the standard size case. It fits conveniently under the Mac. It's a SCSI drive which means it connects to the SCSI port on the back of the Mac Plus. This makes it considerably faster than drives like the HD 20 which connect to the slower serial port.

Speed

MacFast was the fastest of the disks I used. Loading and saving short files seemed almost instant. It's reported to be four to six times faster than the Macintosh HD 20.

Noise

MacFast was also the quietest external drive. It makes virtually no noise when it accesses the drive and its fan makes an acceptable amount of noise. It's not at all bothersome sitting close by under the Mac.

Flexibility

MacFast is reasonably flexible. You can partition it into up to five volumes. This way you can have both HFS and MFS volumes. If you don't need MFS volumes, you can also make the entire disk one HFS volume or if several people share the disk, you could give each his or her own volume. There is no utility available to password protect the volumes.

You can daisy chain the MacFast with up to six other SCSI devices. Peripheral Land has a tape backup system that connects to the drive, but this was not available for testing.

Reliability

While using this disk I did get several "unrecoverable disk error" messages. I think what happened was that a sector went bad and when the application tried to write there, it couldn't. I backed up the disk, then reformatted, and all was well. What the formatting program does is pick up any damaged sectors and mark them as unusable.

Utilities

MacFast comes with the usual installation and format utilities. Because the disk does not automatically move the head to a non-data area at power down, it also comes with a utility called Park It to be used before moving the disk.

A backup utility is provided but is difficult to use. The documentation is poor. The program appeared to have all the required parts — to back up the entire disk or selected files or folders — but I couldn't get it to work. No one at Peripheral Land was of much help. It appears to have been written by an outside vendor, and no one at Peripheral Land knows much about it. As yet there are no other utilities, but Peripheral Land does plan to provide a disk test utility.

Support

Support falls into the "don't count on it" category. They do have a technician who will help when you call, but several times he said he'd consult with the engineers and call back and this never happened.

Overall

The MacFast is a good, fast, quiet drive. It's reasonably priced, reliable, and a good choice if you don't need fancy utilities and have an

alternate backup program. If you don't have special cases, you won't need much support and can most likely get what you need from Peripheral Land's technician and your dealer.

MDIDEAS WITH ADON TAPE

The MDIdeas drive fits nicely under the Mac but is longer than other drives to accommodate both the disk drive and the tape drive (see Figure 2-3).

The tape is a 40 MB 3M mini-date cartridge that fits into the 3-½" tape drive. The tape is slow compared to the disk, but is reasonably fast for tape. And the Adon tape driver is amazing. Typically you store a mirror image of the entire disk on tape, and the only way to retrieve information from the tape is to restore the entire tape to the disk. The Adon driver doesn't do this. It saves files on the tape just as if they were on disk, so you can use the Finder to locate a particular file or folder and drag it to the hard disk just as you would from a floppy or another hard disk.

Speed

The MDIdeas drive is reasonably fast. It's not quite as fast as MacFast, but it's much faster than the Macintosh HD 20. And I thought it was fast enough to make working with it painless.

The tape is very slow compared to a disk, particularly a SCSI hard disk. It takes a few minutes to rewind when you insert the tape, saving to it and restoring from it is slow, but it is no slower than other tape backup systems I've used. And being able to use the tape like a disk — to use the Finder to locate just the file you need — is worth the wait.

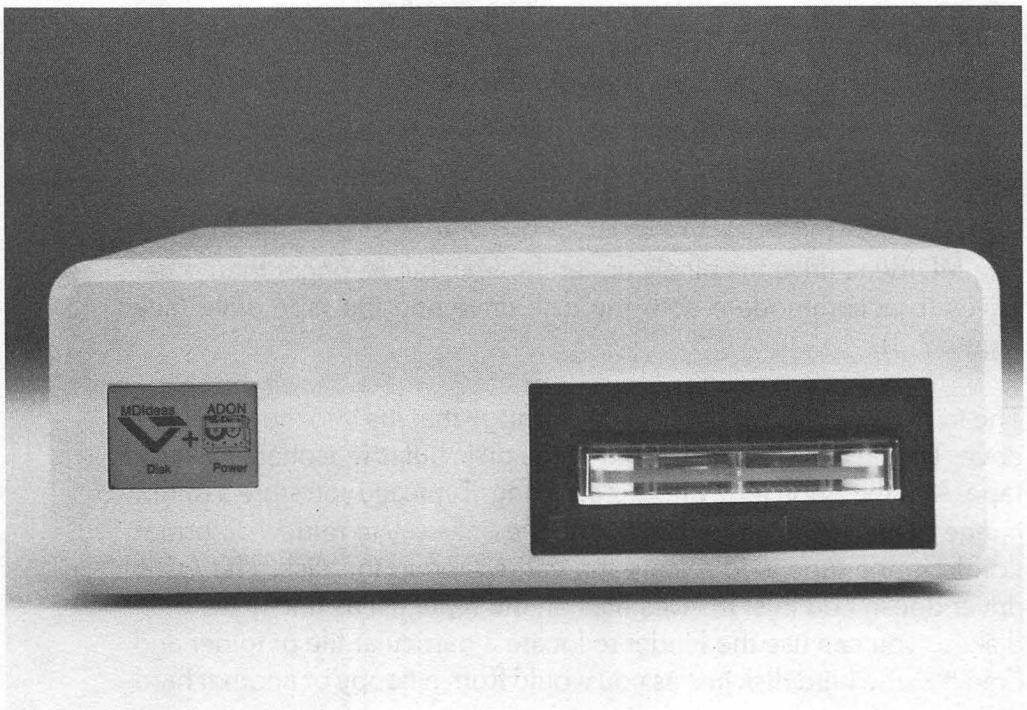


Figure 2-3. Photo of drive.

Noise

The drive is fairly quiet. It has a fan that even my partner does not find offensive. And when you access the drive, it makes a comfortable, low-pitched "chortle". The tape, however, is noisy. That's just the state of tape technology now. I didn't find the tape noise to be a problem because it wasn't something I had to listen to while working. Typically I would only use it at the end of the day just before shutting down.

Flexibility

The MDIdeas drive cannot be partitioned. It is a single HFS partition. Like the MacFast this is a SCSI device so it can be daisy-chained with up to six other SCSI devices such as additional disks, scanners, or tape backup systems.

Utilities

The disk comes with a tape driver that lets you use the tape like a disk and with two additional programs. One is a mirror image program to backup an entire disk and the other is an incremental backup program. The mirror image currently backs up from disk to tape at the rate of 1 megabyte per minute. Information sent to the tape is file addressable which means you can use the Finder to locate a specific file or folder and restore only that file or folder. The incremental backup program allows you to do selective backup and restore by folder, by date, or to backup everything that's changed since the last backup.

Adon has plans to release an additional utility that will let the backup program run in the background. Their existing tape backup system gives you few excuses for not backing up; the pain of swapping floppies is eliminated. But if you could backup each night and at the same time take care of all those last minute tasks you need the Mac for, you'd have absolutely no excuse.

Reliability

The MDIdeas drive passed the ultimate test. It was riding in the back seat of my car when I came to a sudden stop. It slipped off the back seat on to the floor. I plugged it in expecting the worst, but in fact had no problems. For the two months I had the drive, I never had a read or write problem or a bad sector requiring me to reformat.

Support

Because I never had a problem, I didn't use either MDIdeas or Adon's support. I did have information questions that were answered promptly. Adon says that they do provide technical support for both the disk and tape drive and are happy to answer questions. If the problem requires testing, they claim they'll do that and get back to you with a solution.

Overall

This drive was my favorite. It was acceptably fast and quiet and incredibly reliable. The tape makes life easy for hard disk users. To have the tape as part of the hard disk package and to be able to use it without any special training just as you would another disk was a great boon. It is expensive ($3000 for the 30 MB disk with the 40 MB tape), but you can buy the disk and tape separately. A friend who calls himself a "disk slob" has an MDIdeas drive with the Adon tape. He loves the fact that with this disk and tape he can totally eliminate floppies. However, at last visit I observed he's now become a "tape slob" — he had six 40 MB tapes next to his Mac!

CHAPTER 3

HFS and the New Finder

THE MFS

Original 1984 Macintoshes used 400K floppy disks that were organized with the Macintosh Filing System or MFS. Each disk could hold up to 64 files which was typically more than enough for a 400K disk. With the MFS you could put files into folders. This got rid of clutter on the desktop and made it more convenient to work with a disk with a large number of files. You could set up the desktop so you could look into a folder to find what you needed instead of scrolling through a large number of icons. Although the folders made the desktop look organized, they did not physically organize the information on the disk. When you started an application and chose the Open command, the dialog box would list all documents on the disk created with that application rather than only listing those in the currently selected

folder. And you could not have two documents with the same name even if they were located in different folders.

The first generation of Mac hard disks used the MFS, but it was not very effective for a hard disk with ten or more MB's of storage space. In the first place, most hard disks have enough disk space to store well over 64 files. If you tried to add a 65th file, even if there was room on the disk, you'd see the message: "The destination directory is already full. OK?". It was definitely not OK, but there was nothing to do be done. There was no more room in the directory to store titles of files and information about where they're stored on the disk.

If the disk driver software simply increased the directory space, allowing say 1,200 files, with the MFS organization this would be more files than the Finder could track effectively. Imagine each time you opened an application having the standard file dialog box list over a thousand files for you to scroll through and choose from. It would be slow and inconvenient. And recreating the desktop with an extremely large number of files and folders when you quit an application would seem interminable. Clearly the MFS was not the optimal organization for disks with the capacity of today's hard disks.

INTERIM ORGANIZATION PLANS

The second generation of hard disks, ones like HyperDrive, adapted the MFS to work with their hard disk systems. They carried the file and folder metaphor one step further and partitioned the disk into drawers or volumes. The drawers looked like subdirectories, but the system looked at each drawer as a separate disk or volume. When you needed information or applications in a particular drawer, you first had to open or mount that drawer. You could have several drawers open at once, but if too many were open, entering and exiting applications slowed down considerably.

Although this solution was a vast improvement over the straight MFS for managing hard disks, it had some limitations. Files in drawers could be put into folders, but as with the plain MFS, these folders were only cosmetic. If you had a large number of documents for a particular application, for example Word, when you opened the Word drawer and started Word, you'd see a list of all your Word documents in the standard file dialog box. You could, however, use separate drawers for different types of documents, and as long as all the drawers were open or mounted you'd need one copy of the application and could load documents into that application from any drawer.

Also, moving things from drawer to drawer was a two-step process. You couldn't carry the drawer metaphor one step further and move files and folders from one drawer to another as you can with manila folders and paper files. Instead you had to copy the folder or file to the new drawer and then delete it from its original drawer.

THE HFS

What Is It

Finally, when Apple delivered their own hard drive, the HD 20, they produced a system to really manage the number of files typically stored on a 20 MB disk. It was called the HFS or hierarchical file system. The HFS can handle a large number of files — well over 1,000, and it manages the files by letting you arrange them in nested subdirectories. It uses folders for the subdirectories, and what you do to organize information and applications is put folders inside folders. Each lower rank is nested in the folder above it. You can have as many levels as you need (see Figure 3-1).

The major advantage of the HFS over the MFS is that the Finder recognizes folders. The standard file dialog box now only displays files in the current folder, and lets you move up, down, or across the

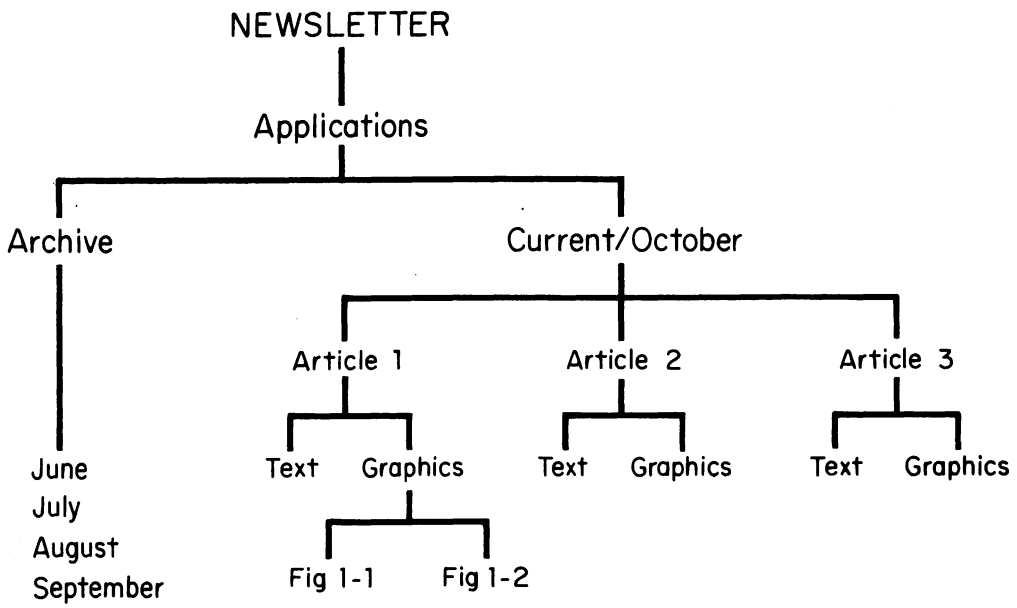


Figure 3-1. Example of nested folders.

hierarchy from folder to folder. To find a document or folder rather than searching through an exhaustive list of every file on the disk, you work your way through a series of logically nested folders.

The HFS was implemented in the System 3.1 and Finder 5.1 software. This software patched the old system in ROM. Then with the Mac Plus and the new ROM, the HFS was incorporated into ROM. You can run HFS and use it adequately on a Mac without the ROM upgrade, but the new ROM makes it work considerably faster.

Changes You Actually See

When you start your Mac with System 3.1 (or higher) and Finder 5.1 (or higher) on a hard disk or an 800K floppy, the initial desktop looks much the same as it did with the MFS. What you see is the top level of the hierarchy symbolized by a disk icon representing the current startup disk. You'll see either a hard disk icon such as the Apple HD 20 or an icon representing an 800K floppy disk (see Figure 3-2).

When you open this icon you see the files and folders in the desktop or root directory (see Figure 3-3).

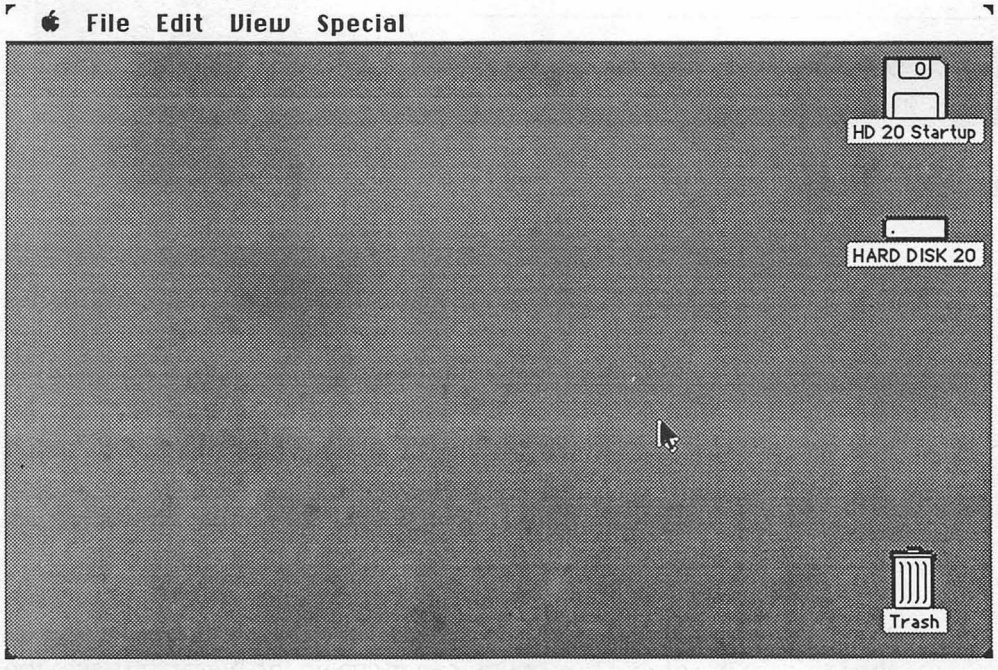


Figure 3-2. Two icons — HD 20 icon and 800K startup floppy icon.

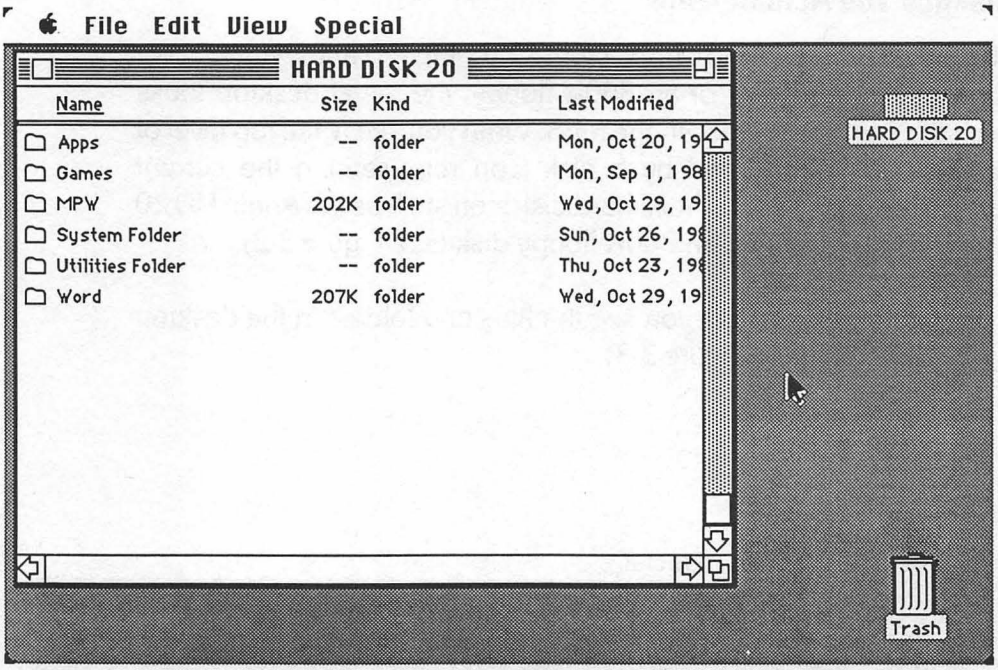


Figure 3-3. Desktop/Root directory.

But unlike the MFS these are not the only folders. You can open a folder on the desktop, find a folder nested inside of it, open that folder, and continue on until you reach the bottom folder (see Figure 3-4).

You can also make your way through the hierarchy and quickly reach any folder with the standard file dialog box. The dialog box instead of listing all documents for the application you're running, only lists folders and files in the current folder. And the window that displays the files and folders is larger and includes an auxiliary menu for folders. You can move to any folder further down the hierarchy by clicking on the folder in the dialog box or you can move up the hierarchy using the folder menu (see Figure 3-5).

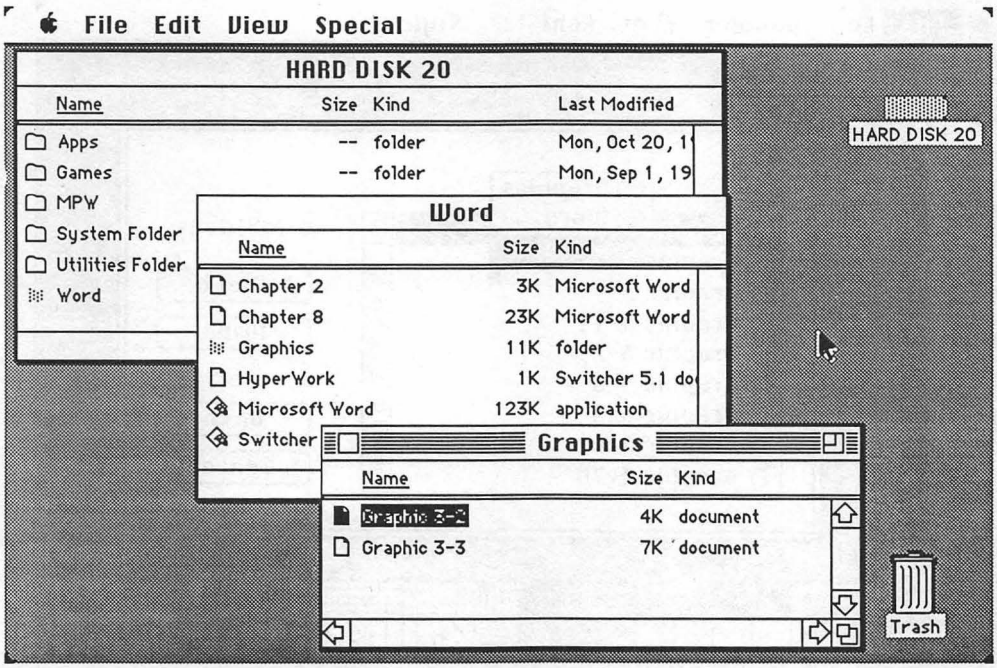


Figure 3-4. Nested folders (3 deep).

System 3+ and Finder 5+ automatically run HFS if you have a hard disk or an 800K floppy disk but will consider applications you run from 400K floppies MFS applications and won't recognize nested folders. If you have sharp eyes you can confirm an HFS or MFS disk. HFS desktop windows have an extra pixel on the extreme left between the double lines near the top of the window. MFS disks do not (see Figure 3-6). Chapter 4 has instructions for using applications that run under MFS with HFS and a hard disk.

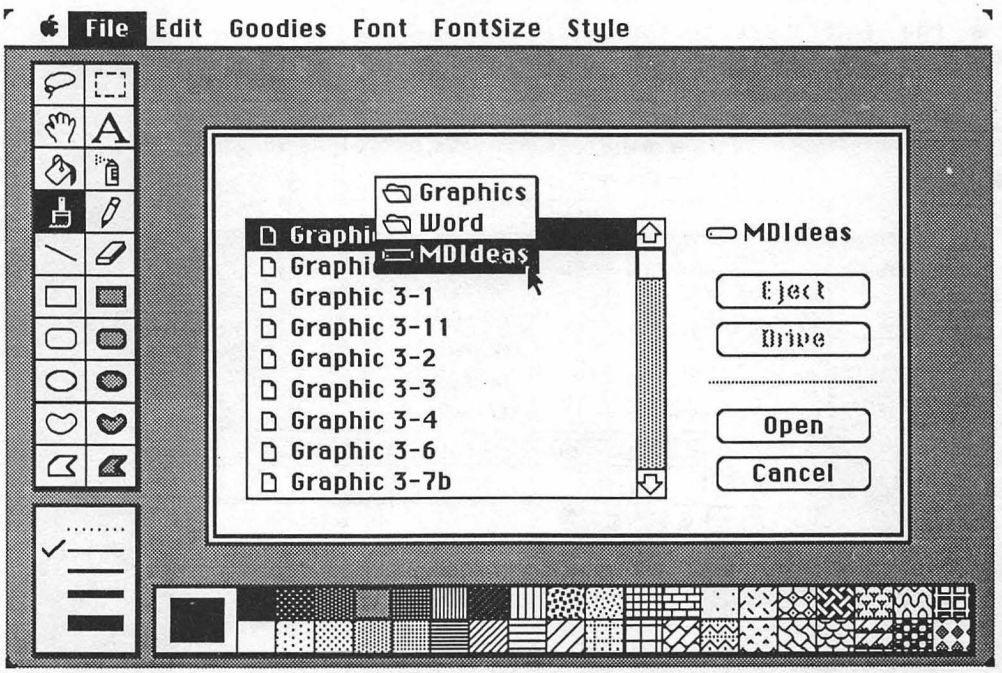


Figure 3-5. Folder menu pulled down.

MOVING THROUGH THE HIERARCHY

From the root desktop it is easy to move down through the folders in the hierarchy. As with MFS, simply double-clicking a folder opens it. From there you can proceed to open the next level by double-clicking on a folder there and continue on down. It is obviously easier to find what you need if folders are given clear, descriptive names and are nested logically.

As mentioned earlier you can also move through the hierarchy from the standard open file dialog box. The dialog box displays the names

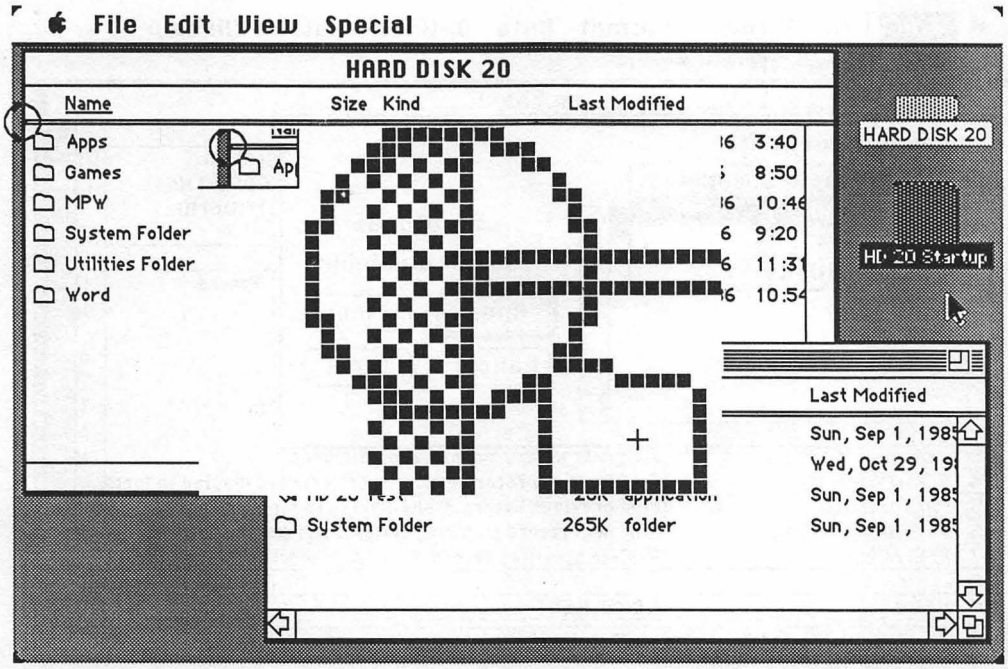


Figure 3-6. Window showing extra pixel between double lines in an HFS disk.

of files in the current folder. To move down the hierarchy, simply double-click the folder name. The folder you clicked on will open and its contents will be displayed in the open file dialog box. To move to up in the hierarchy, pull down the folder menu. Select the folder of your choice and release the mouse button. The open file dialog box will now display the folders and files in the new folder and the folder name changes to reflect the move (see Figure 3-7a,b).

Obviously there is a tradeoff between the number of folders you'll want and the number of documents in each folder. If you have many folders many levels deep, it will take a long time to locate documents.

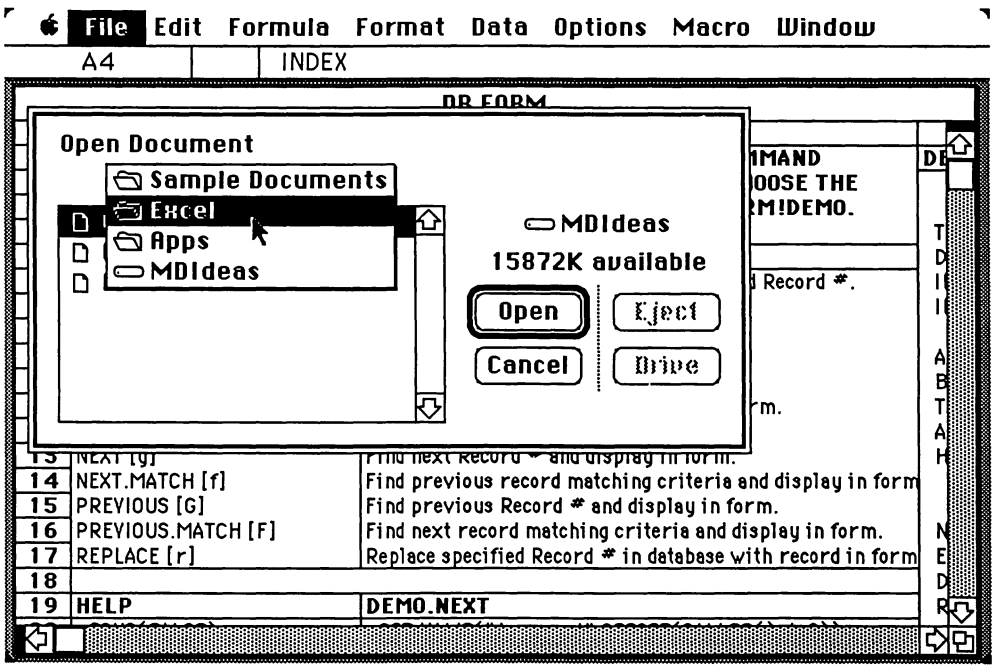


Figure 3-7a. Choosing new folder.

If you have only a few folders, each will contain quite a few documents and you'll have to scroll through a long list to find what you need. The challenge is setting up an organization plan that works for the kind of work you do. Chapter 4 shows several types of organizations and gives you guidelines for any plan you choose.

Note that the standard file dialog box lists folders and files alphabetically. To avoid excessive scrolling, give files and folders you use most often names near the start of the alphabet or preface the name with a number. Numbers will precede letters in the list (see Figure 3-8).

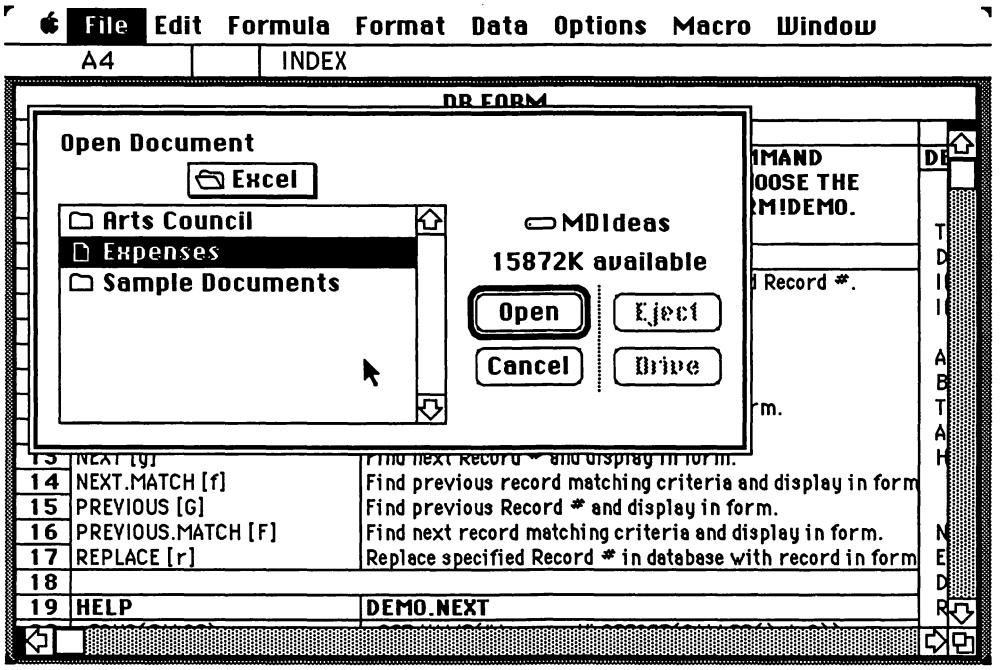


Figure 3-7b. Result of choosing new folder.

You can start an application from a folder located any place in the hierarchy. And as before, you can start an application by double-clicking a document icon. But this document can be in any folder — it does not have to be in the same folder as the application or in a folder in any way connected to the folder where the application resides. No matter where the application is located, Finder (aptly named) will find it, start it, and load the document (see Figure 3-9).

In addition Finder "remembers" the folder the document came from. When you choose Save As . . . , it proposes saving the current document

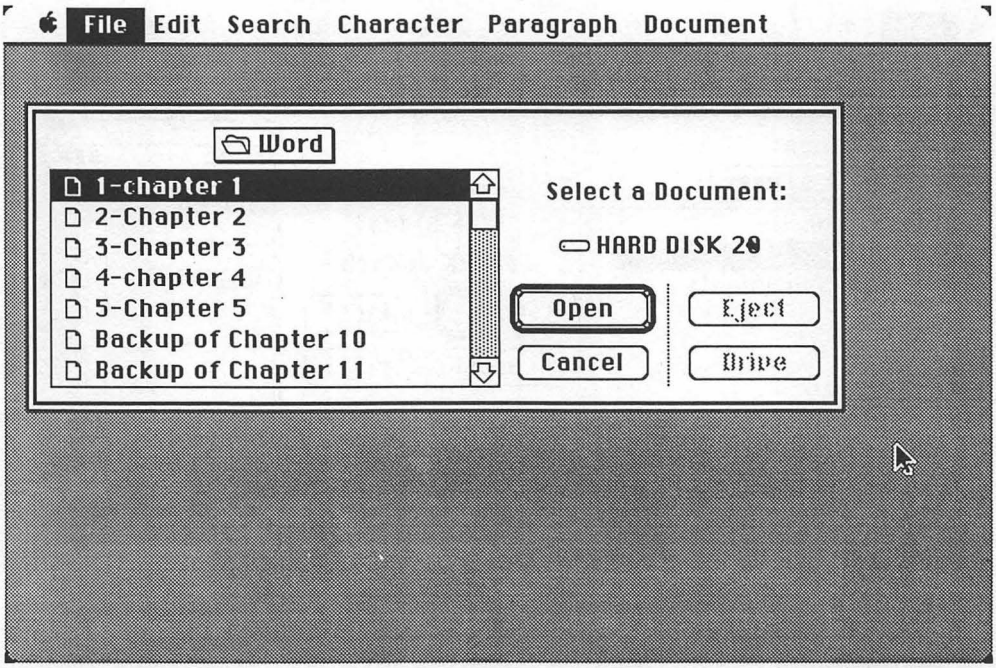


Figure 3-8. List beginning with numbered folders.

in its original folder. This saves time. You only need to move through the folder hierarchy once to locate the original document; you can start the application there and unless you want to change its location, choosing Save will save it in the folder you found it in.

When you move a file from one disk to another by dragging its icon, the system actually makes a copy of the file. But with HFS and the hard disk when you move a file, for example from one folder to another, instead of copying the file, it is actually moved. This more closely resembles what you will want most of the time — you no longer have to copy and delete when what you want is to move. (If

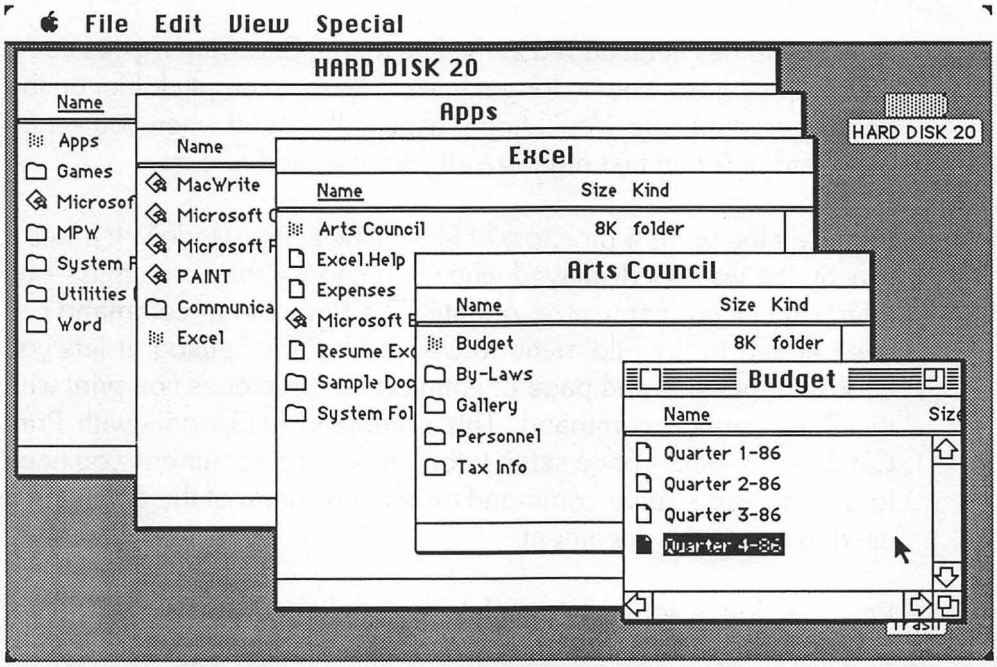


Figure 3-9. List beginning with numbered documents.

you do really want a bonafide copy, you can, of course, use the Duplicate command on the File menu.)

NEW FINDER COMMANDS FOR HARD DISK SUPPORT

Finder 5+ has added some new commands for hard disk support. Four new commands are on the File menu and one is on the Special menu.

The File Menu

New Folder has been added to the File menu. Choosing it gives you a new empty folder. You no longer have to keep an empty folder on the desktop to duplicate. You'll find it especially useful when you set up your hard disk and first organize all your files and folders.

Print Catalog prints a directory of files in the active window. It prints it exactly the way it's displayed when you choose the command — by icon, small icon, name size, or date. The Page Setup command has been added to the File menu to work with Print Catalog. It lets you choose paper size and page orientation for directories you print with the Print Catalog command. This command works only with Print Catalog. To specify page setup information for a document, you need to use the Page Setup command on the File menu of the application used to create the document.

Print Catalog is somewhat useful. You could go through each folder on your hard disk, print a directory of each, and end up with a complete directory listing, but if your disk had a large number of files, this would take a long time and it would still be difficult to see the big picture. If you have a large number of files and occasionally have difficulty locating one or if several people share a disk, you may find a utility such as Disk Ranger (Mainstay Computing, Agoura Hills, CA, $49.95) worth the investment. It prints a complete catalog of all files on your hard drive, shows which folder they're located in as well as the size, and type of file (see Figure 3-10).

You can also use it for floppy disks — to print a directory of files on the disk and to print labels. The label-making feature could be useful for labels for backup disks.

The last new command on the File menu is Put Away. It's what some of us have always wanted — someone to pick up after us. It returns files and folders from the desktop or trash back to where they came

Figure 3-10. Disk Ranger catalog.

File Name	Att	Modified	Type	Creator	Size	Volume	Folder
1-chapter 1		10/28/86	WDBN	WORD	20608	HARD DISK	:Word
2-Chapter 2		10/21/86	WDBN	WORD	2432	HARD DISK	:Word
3-Chapter 3		9/17/86	WDBN	WORD	18560	HARD DISK	:Word
4-chapter 4		9/22/86	WDBN	WORD	19712	HARD DISK	:Word
5-Chapter 5		10/2/86	WDBN	WORD	13312	HARD DISK	:Word
5th Symphony	L	12/31/84	CWMF	CWMP	2426	HARD DISK	:Music
Aase	L	12/31/84	CWMF	CWMP	925	HARD DISK	:Music
Alto Sax.	L	12/14/84	CWIF	CWIM	814	HARD DISK	:Single Reed
Alto Voice	L	12/13/84	CWIF	CWIM	814	HARD DISK	:Vocal
Amuseum		1/16/85	CWMF	CWMP	493	HARD DISK	:Games
Anitra's Dance	L	12/31/84	CWMF	CWMP	2816	HARD DISK	:Music
AppleTalk		5/15/86	????	????	6144	HARD DISK	:Libraries
Asm		5/19/86	MPST	MPS	181669	HARD DISK	:Tools
Baby Elephant	L	12/31/84	CWMF	CWMP	922	HARD DISK	:Music
Backup		10/26/86	APPL	????	8307	HARD DISK	:JimT
Backup of Chapter 10		10/14/86	WDBN	WORD	512	HARD DISK	:Word
Backup of Chapter 11		10/23/86	WDBN	WORD	9984	HARD DISK	:Word
Backup of Chapter 5		10/1/86	WDBN	WORD	13184	HARD DISK	:Word
Backup of Chapter 6		10/1/86	WDBN	WORD	4608	HARD DISK	:Word
Backup of Chapter 8		10/20/86	WDBN	WORD	512	HARD DISK	:Word
Backup of Chapter 9		10/9/86	WDBN	WORD	12032	HARD DISK	:Word
Backup of Standard 6		10/15/86	GLOS	WORD	640	HARD DISK	:Word
backup.p		10/26/86	TEXT	MPS	25462	HARD DISK	:JimT
Backup.p.o		10/26/86	OBJ	MPS	6144	HARD DISK	:JimT
backup.r		10/20/86	TEXT	MPS	6008	HARD DISK	:JimT
Backup.r.o		10/20/86	APPL	????	2166	HARD DISK	:JimT
backup.s		10/26/86			124416	HARD DISK	:JimT
backupmake		10/26/86	TEXT	MPS	667	HARD DISK	:JimT
bak		10/7/86	TEXT	MPS	1327	HARD DISK	:JimT
Barbara Allen	L	12/31/84	CWMF	CWMP	1672	HARD DISK	:Music
Bass Clarinet	L	12/15/84	CWIF	CWIM	814	HARD DISK	:Single Reed
Bass Pizz.	L	12/13/84	CWIF	CWIM	814	HARD DISK	:Plucked
Bass Voice	L	12/13/84	CWIF	CWIM	814	HARD DISK	:Vocal
Bassoon	L	12/13/84	CWIF	CWIM	814	HARD DISK	:Double Reed
Bee	L	12/5/84	CWIF	CWIM	814	HARD DISK	:Original
Blue Danube	L	12/31/84	CWMF	CWMP	14792	HARD DISK	:Music
Bumble Bee	L	12/31/84	CWMF	CWMP	3287	HARD DISK	:Music
Bydlo	L	12/31/84	CWMF	CWMP	2192	HARD DISK	:Music
Canon		5/19/86	MPST	MPS	17795	HARD DISK	:Tools

```
Canon.Dict              5/16/86 TEXT MPS   17340 HARD DISK :Tools
Cello              L   12/13/84 CWIF CWIM    814 HARD DISK :Bowed
Cello Pizz.        L   12/13/84 CWIF CWIM    814 HARD DISK :Plucked
Chapter 10 Notes        10/14/86 WDBN WORD  10240 HARD DISK :Word
Chapter 11              10/24/86 WDBN WORD  13056 HARD DISK :Word
Chapter 6               10/6/86 WDBN WORD   20864 HARD DISK :Word
Chapter 7               10/21/86 WDBN WORD  21120 HARD DISK :Word
Chapter 8               10/23/86 WDBN WORD  22656 HARD DISK :Word
Chapter 9               10/10/86 WDBN WORD  12800 HARD DISK :Word
Clarinet           L   12/13/84 CWIF CWIM    814 HARD DISK :Single Reed
Clipboard File          10/29/86 CLIP MACS      8 HARD DISK :System Folder
Clipboard File           1/4/86 CLIP MACS      0 HARD DISK :System Folder
Closed Flute       L   12/15/84 CWIF CWIM    814 HARD DISK :Pipe Organ
COMMANDS                 9/24/85 XLPG XCEL   6364 HARD DISK :Sample Docume
Compare                  5/18/86 MPST MPS   35926 HARD DISK :Tools
Comparison          B    1/8/86 APPL Cmpr   14410 HARD DISK :Tempo1.1
Cornet             L   12/14/84 CWIF CWIM    814 HARD DISK :Brass
Cornopean          L   12/11/84 CWIF CWIM    814 HARD DISK :Pipe Organ
Count                    5/19/86 MPST MPS   14865 HARD DISK :Tools
Courante           L   12/31/84 CWMF CWMP    1301 HARD DISK :Music
cpy.p                   10/13/86 TEXT MPS    1968 HARD DISK :JimT
```

from on the disk. This can be useful if you've moved applications, for example, out of folders, then backed up the documents in the folders onto floppy disks. Instead of moving the applications back into the folders by hand, you could simply choose Put Away and have the Finder do the work for you.

The Close All command previously on the File menu for closing windows is gone. This is unfortunate because it would be a handy way to clean up the desktop before starting an application. If you start with a clean desktop, returning to the Finder when you quit does not take as long because there is not that much for the Finder to reconstruct.

The View Menu

The View menu has added one new command, Small Icon. It's there so you can still display a directory in icon form, if you wish, but you'll be able to see more of them in a window at once (see Figure 3-11).

The Special Menu

The Special menu has one new command, Use MiniFinder . . .. You can find instructions and tips for using the MiniFinder in chapter 6.

Shut Down, a command on earlier Finders, works differently than it did before if you have a hard disk and the new ROM. Before, if you

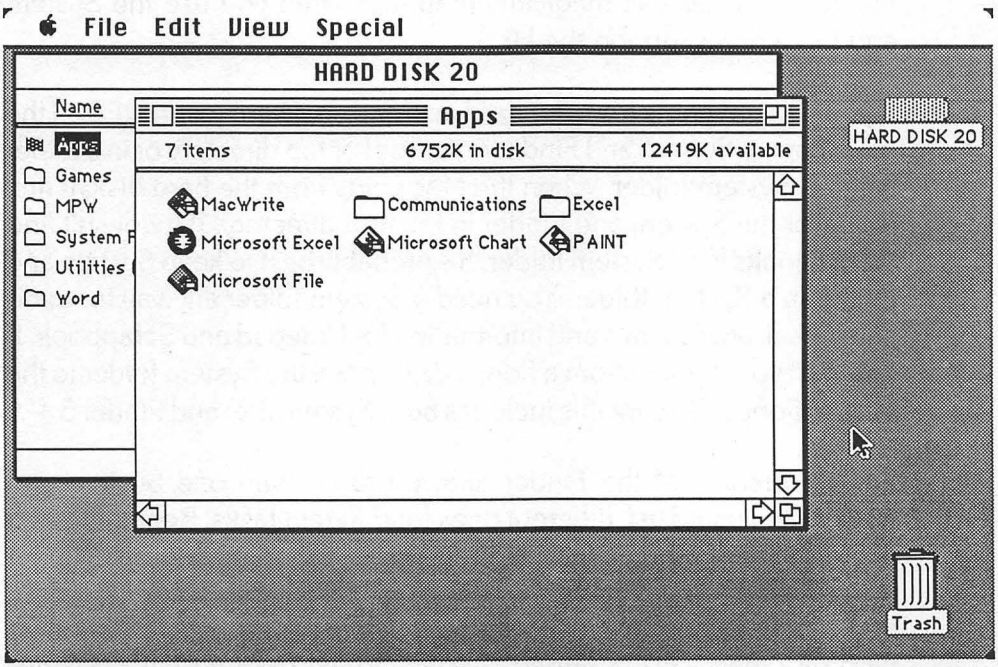


Figure 3-11. Small icons.

chose Shut Down, necessary information was saved, floppy disks were ejected, and the screen was blanked. Now with a hard disk that starts the system, Shut Down saves necessary information, ejects any floppy disks,and restarts the Mac. I have yet to figure out a good reason to use it and let it restart the system. However, if you have quick hands, you can use it like you used to — as soon as the screen blanks and before it restarts, reach back and turn off your Mac.

SETTING UP

Systems earlier than System 3+ and Finder 5+ do not recognize HFS. If you try to run something you created with HFS on a Mac running a System and Finder prior to 3+ and 5+, you'll find that Finder only "sees" files and folders in the topmost directory. Files and folders in subdirectories will magically reappear when you use the System and Finder that support the HFS.

If your system starts from the hard disk and you want to run HFS all the time, put the System and Finder in the root or top directory or in a folder labeled System folder. When the Mac starts from the hard disk, it first looks for the System and Finder in the root directory. If it doesn't find them it looks for a System folder. It's probably best to keep System and Finder in a System folder. You need a System folder anyway for dedicated desk accessories and information for Notepad and Scrapbook. If you start your system from a floppy disk, move the System folder to the startup floppy. Be sure this includes both System 3+ and Finder 5+.

Earlier versions of the Finder allowed more than one System and Finder. You could use different ones for different tasks. Beginning with Finder 5.1 more than one System folder is no longer allowed. Finder takes the System from the root directory, calls it the "blessed folder", and uses it exclusively. But with so much more space available on a hard disk than a 400K floppy — space for up to 16 desk accessories and as many fonts as you can find — it is no longer as necessary to

have different systems for different tasks. You can most likely put everything you need for all your work in the "blessed" System folder.

Some hard disks allow you to partition the disk into separate volumes. Because the volumes are treated as separate disks they can have separate System folders. If this is important to you, you may want to consider a hard disk that allows partitions. For information on specific disks, see chapter 2.

The HFS is a good, easy-to-use system for managing 20 or more megabytes. It's better than MFS because it lets you trace through logically nested folders each with a workable number of documents and folders to find what you need instead of scrolling through a list of several hundred documents. You only need one copy of an application; no matter how deeply it is buried, the system will find it. It's easier to use than partitioning the disk into drawers that have to be separately opened and closed. With HFS managing a hard disk, everything is at your fingertips ready to use.

But first you need to copy in the applications and documents you use. Before you start copying, first consider how to use the HFS and organize the tasks you do in the space available on the hard disk. Obviously the goal is to make your work as efficient as possible. Chapter 4 has specific organization plans. Read it first; then choose and modify a plan to suit the way you work.

CHAPTER 4

Organizing Your Disk

BASIC GUIDELINES

Before you rush out and madly copy everything you now have on floppies onto your hard disk, wait. Consider this analogy — what if last year when you received a three-drawer filing cabinet, you neglected to put in any hanging folders or tabs. Whenever you wanted something filed, you simply threw it into the first available drawer. Imagine the difficulty of finding those budget figures you "filed" three months ago and need for a meeting in ten minutes. A 20 MB hard disk, like a filing cabinet, can store as much as 10,000 pages of information. And a random organization of folders and files on a hard disk makes about as much sense as it does in a filing cabinet.

You most likely bought a hard disk to improve performance and convenience. Without a careful organization plan, you'll get neither. If

you don't take the time to put a file into the proper folder with a descriptive name, you'll end up spending valuable time playing adventure to relocate it, and that really slows down performance.

A good organization plan balances the number of folders and the number of files in each. With the HFS you're torn between two mutually exclusive needs — to keep folder nesting to a minimum and the number of documents per folder to a minimum. If you nest folders too deeply, you'll spend too much time digging out what you need. And if you put too many documents in a folder, you'll spend too much time scrolling to find a specific document. A simple rule of thumb is to keep no more than five to six documents and one to two folders inside any one folder. This way the entire contents of the folder can be displayed in a standard file dialog box without scrolling the listing inside of the box. If you do need to store more than this, make the folders and documents you use most often come first alphabetically so you can open them without scrolling. If there is one you use most often, make it come first. (You can do this by prefacing its name with the number 1.) The first item in the list will be automatically selected when you choose the Open command (see Figure 4-1).

Basically, you must organize files and folders on your hard disk in a way that suits the way you work. You may find it easier to keep the number of files per folder small and click through a number of folders. Or you may prefer scrolling through a long list of files to get to what you want because you find it useful to see a list of your current projects. The organization you choose may be quite different from the way you worked with MFS. You can, for example, keep all applications in one folder. They don't need to be connected to their documents, and no matter where this application folder is, Finder will locate the application you need when you click on a document. Another consideration is backup. You might consider this when you make an organization plan. You could, for example, create a Daily folder. As you work,

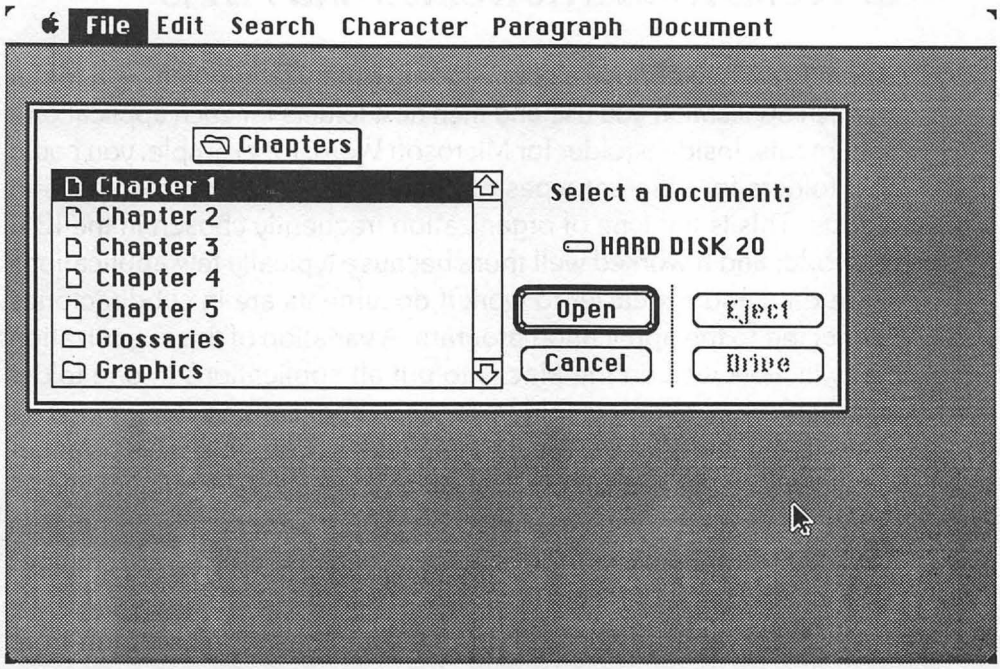


Figure 4-1. All files and folders visible without scrolling.

put completed documents for that day into the folder, and at the end of the day simply drag that folder to a floppy disk for backup.

The sections that follow have some organizational strategies and then examples of three-specific plans. Look over this information and choose the type of plan that will work for you. Sketch out a directory tree on paper. When you have a plan that looks like it will work, then begin — create the folders, give them descriptive names, and then, finally, you can copy in your applications and documents.

ORGANIZATION STRATEGIES AND PLANS

You may wish to choose a conventional organization plan — a folder for each application you use and then nest folders for each application's documents. Inside a folder for Microsoft Word, for example, you could nest folders for different types of documents — letters, memos, and reports. This is the type of organization frequently chosen in the IBM PC world, and it worked well there because typically few applications share data and it's easier to work if documents are in subdirectories connected to the application program. A variation of this organization that will work well on the Mac is to put all applications in one folder and then create separate folders for each application's documents — Excel documents, Word documents, Paint documents. This arrangement can facilitate backup; you can simply back up the document folders of the applications you've worked on. There generally is no need to back up the applications more than once because they do not change.

In the Mac world if your work involves combining information from several applications, you may find it more convenient to organize around tasks rather than applications. If you do a monthly newsletter that contains a spreadsheet and graphics from Excel that you move into Word to add text and then send to the graphics department for electronic pasteup, you may wish to have a folder called Newsletter with the applications you use — Excel, Word, Project, and MacTerminal. You could display the files for the current issue on the desktop and nest an Archive folder to store information from past newsletters by date. This will keep clutter off the desk — you'll only have current graphics projects there. If you prepare financial statements, you could create a Finance folder with Excel, MacTerminal, Switcher, and perhaps MacWrite for sending some explanatory text along with the graphics. If you do presentations, you could have a Presentations folder with Chart, Thunderscan, Paint, and Draw. If you do several tasks and have the room, you might consider allowing yourself the luxury of having more than one copy of a program on your disk.

Another strategy is to group similar applications together. You could have a word processing folder with your word processor, spelling checker, and outliner, a graphics folder with Paint, Draw, Thunderscan, and Chart, and a programming folder with your compilers and utilities.

If your work revolves around four applications or fewer, you might consider this plan. Put the four applications in the root directory. Then drag them out of the disk window onto the actual desktop. Line them up in a column adjacent to the startup disk icon. Then resize the startup disk window so both rows of icons show. At startup you'll have a desktop that looks like Figure 4-2. And the applications you use regularly will be right at your fingertips.

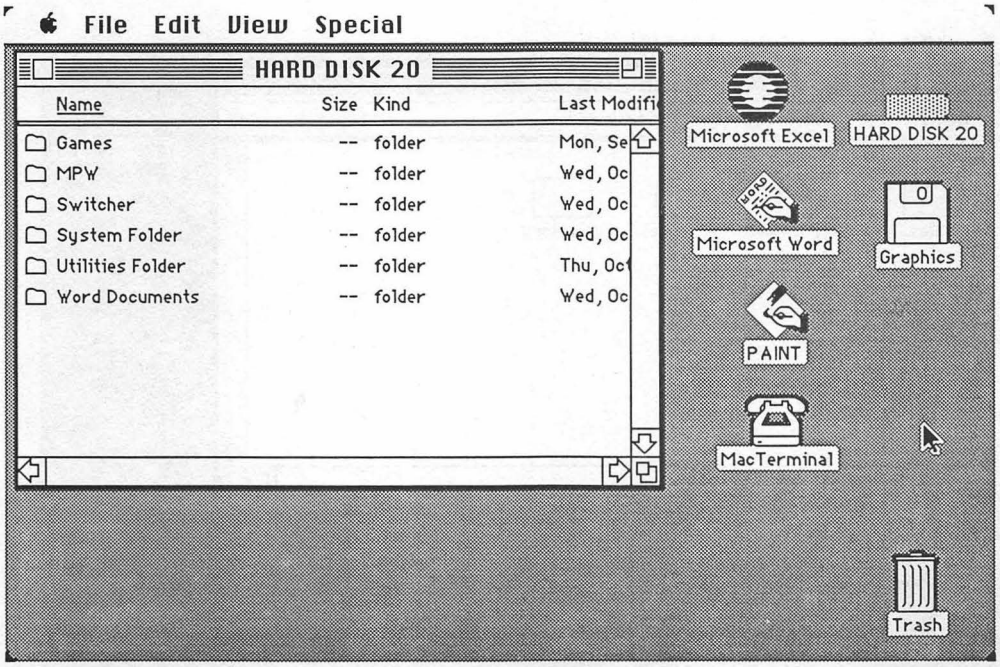


Figure 4-2. Vertical icon arrangement.

Organization by Task

Imagine you're the sales manager for an exotic car dealer. It's your job to track sales of cars, employee sales data, and inventory at three locations. An effective organization for these tasks might look like this. The desktop would have only four folders — System, Sales, Personnel, and Inventory (see Figure 4-3).

You use the Sales folder to track sales and produce monthly sales reports. You use four applications: Excel, MacProject, Word, and MacTerminal. You use MacTerminal to get current sales data from the

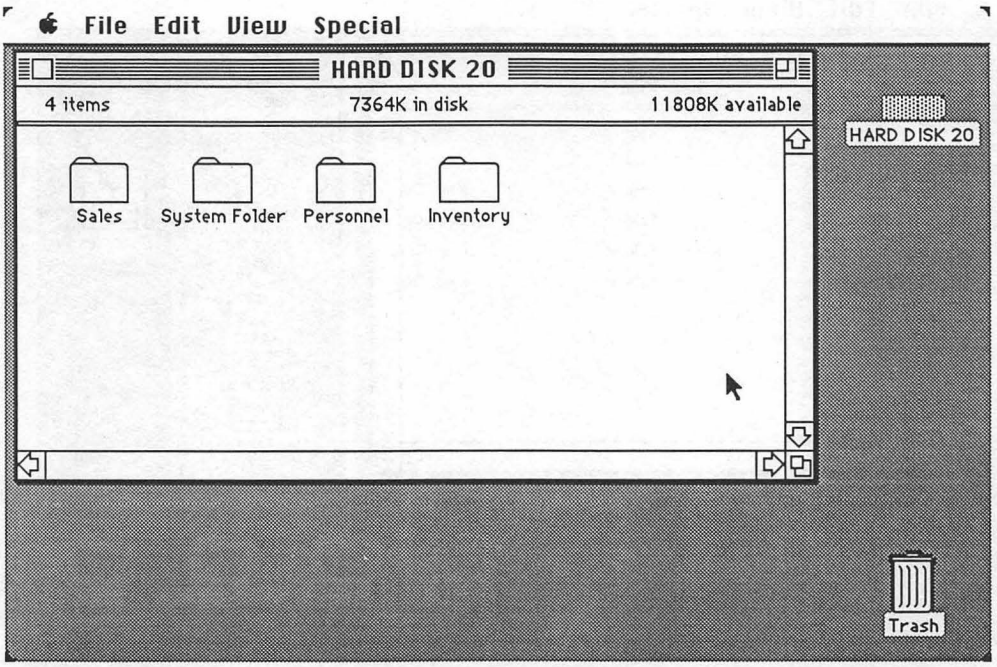


Figure 4-3. Exotic car desktop.

Macs at remote locations. You use the Personnel folder for employee data and to track sales by employee. You use File for the employee datafile and Excel for the sales information. From this information you produce a quarterly report using Word with employee sales data and the top performer for the quarter. You make it look impressive by adding the employee's picture, scanned with Thunderscan and kept in an employee picture file. You use Switcher when you prepare the report making it easier to use information from File, Excel, and Thunderscan with Word. The third major task is inventory. It's your job to track inventory at three locations. You keep inventory information in File and add current information daily from Macs at the other locations using MacTerminal.

Once you devise an organization plan, it's a good idea to draw a diagram to show the contents of each major folder. That way you can get an overview and make sure you have what you need. A diagram of the Personnel folder looks like Figure 4-4.

It contains four major folders — Employees, Reports, Switcher, and Sales Data. Employees has a File document with personnel information about each sales person and a folder called Pictures. If you open Pictures you'll see two folders, A - M and N - Z. These are file folders for pictures of employees created with Thunderscan. If you wish you can incorporate these pictures into the personnel datafiles as well.

The second folder, Reports, contains four documents, reports for each quarter and an Archive folder. The Archive folder is to store completed reports when you need more room on the desktop.

Next is the Switcher folder. It contains only two things — the Switcher application and a Switcher document, Report Time. You can use it to automatically set up the applications you need — File, Word, and Excel — when it's time to do the monthly report.

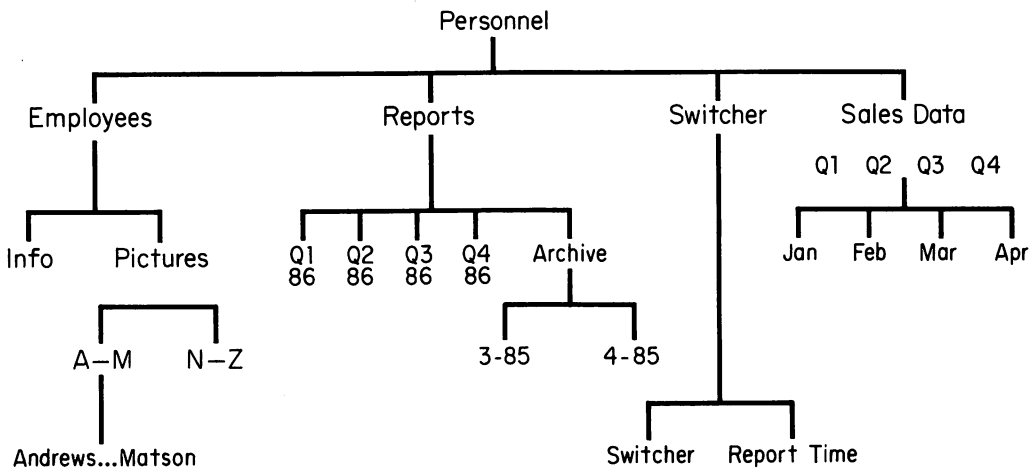


Figure 4-4. Nested personnel folder.

The last folder, Sales Data, contains quarterly folders for monthly Excel sales worksheets. This information was originally created in the Sales section and copied here to be used for the monthly report. Each quarterly folder contains three monthly sales worksheets and one composite quarterly worksheet. The composite worksheet is the information you need when doing the quarterly report. You might want to add an archive folder here and save your quarterly worksheets. An alternative would be to save them in the Archive folder in the Reports directory.

As you can see, it is possible to have folders and documents with the same names — in this example Quarter 1, Quarter 1 . . . — as long as they are in different folders.

You may wonder where the applications are for these tasks. For this type of task organization where several tasks use the same application, rather than keep a separate copy of the application in the folder

of each task that uses it, it's cleaner to store all the apps in a single Application folder. You can store the Application folder anywhere — on the desktop or nested in another folder. Because returning to the desktop is a slow process if you have too many folders in the root directory, you could nest the Applications folder inside of the System folder. But if you use the applications frequently for short tasks and want them accessible without having to open two folders, it might be a better idea to keep the Applications folder on the desktop of the root directory.

Organization by Client or Customer

Imagine yourself a lawyer with a small law practice. You would most likely organize the information on your hard disk by client, and put information into folders much as you would into paper folders. If you had ten active clients, you could keep all ten folders on the desktop. You would name the folders by client, last name first, and set up the Finder to view folders by name. Then you'll see an alphabetical listing of your clients. You'll also want an inactive client folder with perhaps three alphabetical nested folders — A - H, I - R, and S - Z — for filing information on cases you've completed. Your desktop might look like Figure 4-5.

A diagram of a client's folder looks like Figure 4-6.

The Personal folder contains client information — name, address, phone, and a short description of the case. The Time/billing folder contains a record of time spent with the client or working on the case; you can use this for monthly billing. It could also contain an Expense folder. The Correspondence folder contains two nested folders — Letters and Notes. The Notes folder is for information you gather in discussing the case with the client and others. Next is a Pleadings folder for court documents and briefs, the Discovery folder for interrogatories, depositions, and requests for information, and the Expert

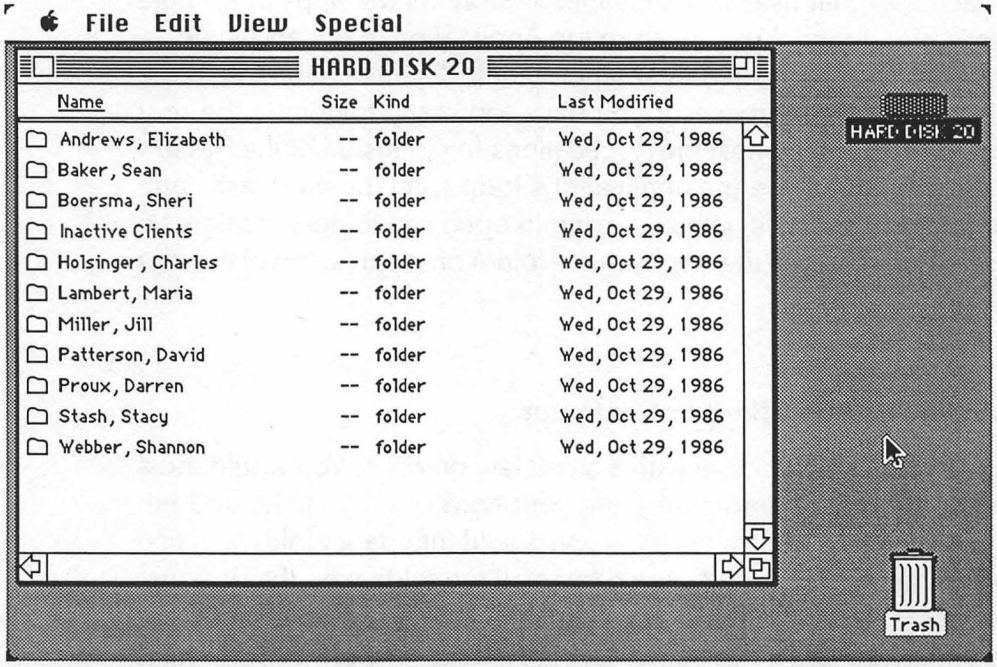


Figure 4-5. Organization by client.

and Research folders for information from experts and about other pertinent cases. You'll choose the types and categories of folders that suit the types of cases you do.

If you have more than ten active clients, you may wish to nest your active clients in an Active Client folder to keep the desktop organized. You may wish to try having Active Client folders loose on the desktop at first, and when finding client folders gets too cumbersome then make an Active Client folder with alphabetical folders nested inside and client folders inside these.

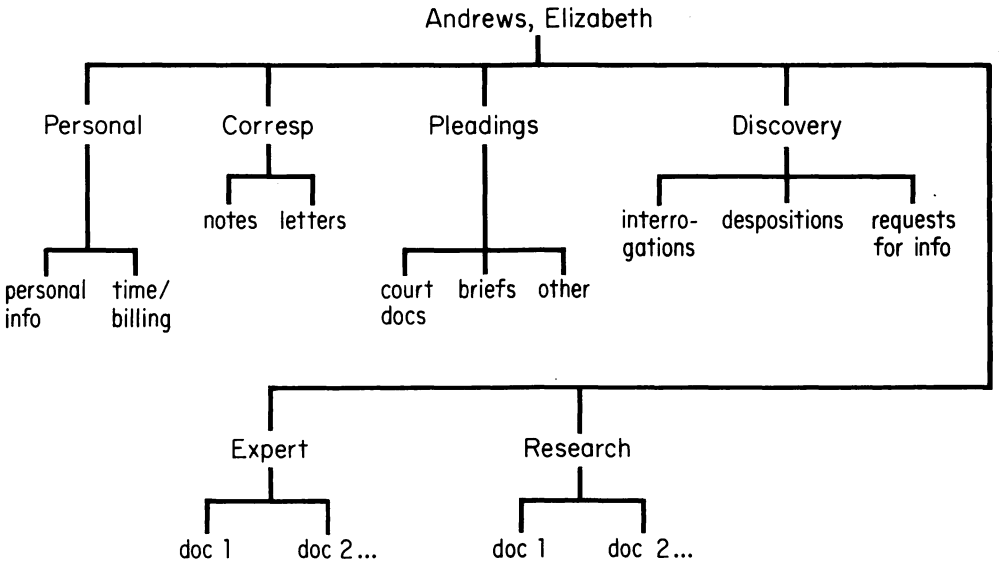


Figure 4-6. Sample client folder.

You can see that it is possible for each client's folder to have nested folders with the same names. If you need similar folders for all clients, you could have a set of empty named folders for new clients ready to be duplicated and named when you are ready to begin entering information for a new client. That way you won't have to create a new nested set for each new client (see Figure 4-7).

You probably will use a word processor like Microsoft Word for most of your work. You might also use a spreadsheet or accounting package for your time/billing information and perhaps an application like File for personal client information. You can keep these applications in an Applications folder inside the System folder or on the desktop. As mentioned earlier, you can still open the applciations by clicking on a document inside a client folder.

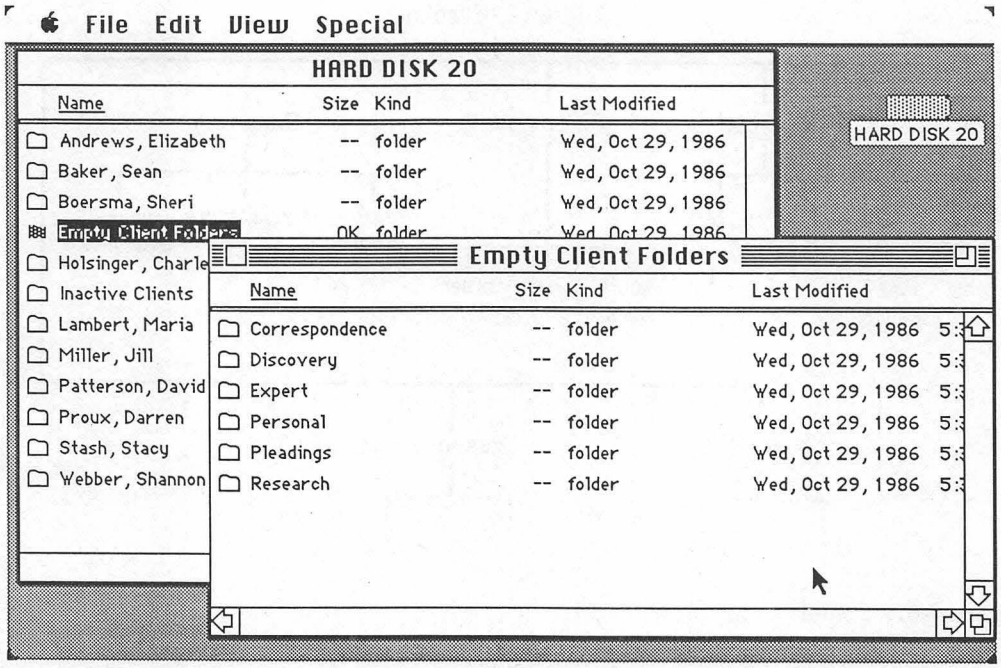


Figure 4-7. Empty client folder ready for new client.

Organization by File

If you work mostly with single applications and don't combine information from several applications, you may prefer a more traditional arrangement. If you use a word processor for letters and reports and a spreadsheet for budgets, you could organize with just two folders (in addition to the System) on the desktop as shown in Figure 4-8. The Letters/Reports folder might look like Figure 4-9 and the Budget folder like Figure 4-10.

When you finished working on a budget, you could simply use the Save As . . . command and store it in the correct month's folder. It's

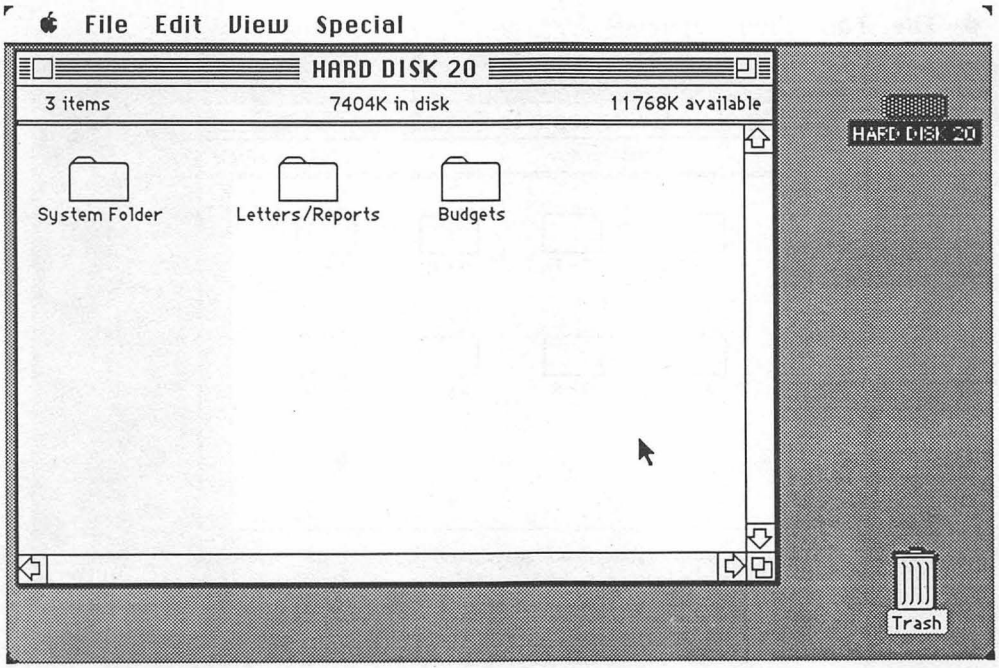


Figure 4-8. Organization by file.

similar to thumbing through folders in a filing cabinet, locating the correct hanging file, and dropping the document in.

SETTING UP

These are only three of the many different types of organization plans. Consider the way you work and choose a plan that will help you work as efficiently as possible. Then sketch out your plan on paper. When you have a written plan that looks reasonable, you're ready to begin creating folders and copying in information.

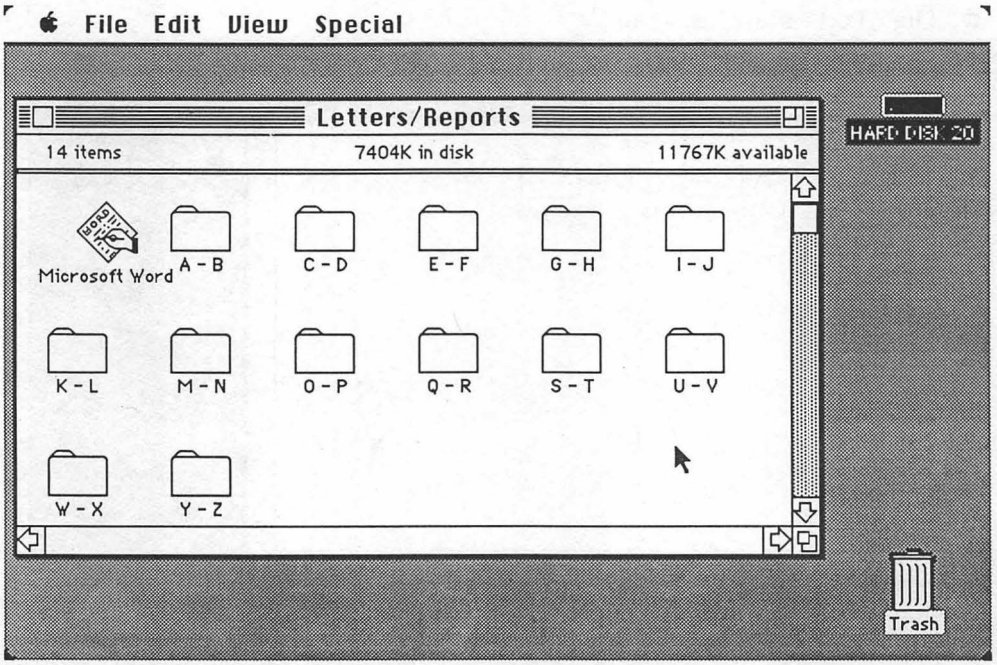


Figure 4-9. Letters/Reports folder.

New Folders

Creating folders is easy. For top level folders open the hard disk window. Then choose the New Folder command from the File menu. You'll get a folder named, not surprisingly, New Folder. Rename it, click on it to open it, and copy in the applications and documents you want to store in that folder. If you want a folder inside this folder, simply choose New Folder when the current folder is the active window, or create a new folder at any level and drag it to the folder you want it nested inside.

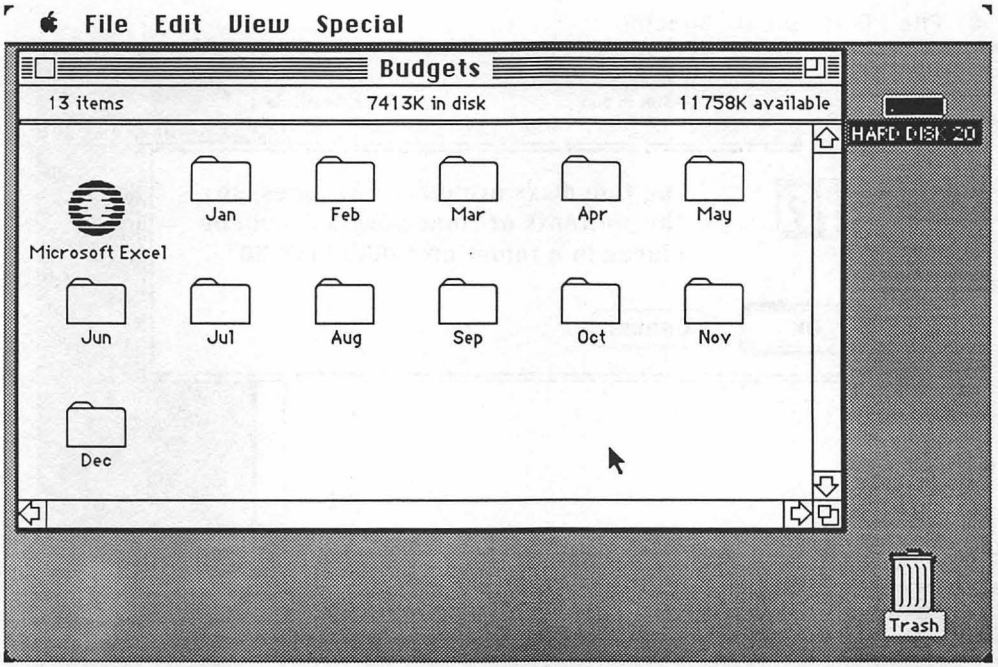


Figure 4-10. Budget folder.

Copying In Information

When you want to copy the entire contents of a floppy disk to your hard disk, you can drag the floppy icon to the hard disk icon or the root window. But if you do this, you'll see the message shown in Figure 4-11.

Click OK and you'll get a folder on the hard disk with the entire contents of the floppy. Once it's there, you can move it and nest it inside whichever folders you want or, if you wish, take it out of the folder.

If you'd rather copy the contents of a disk directly into a nested folder,

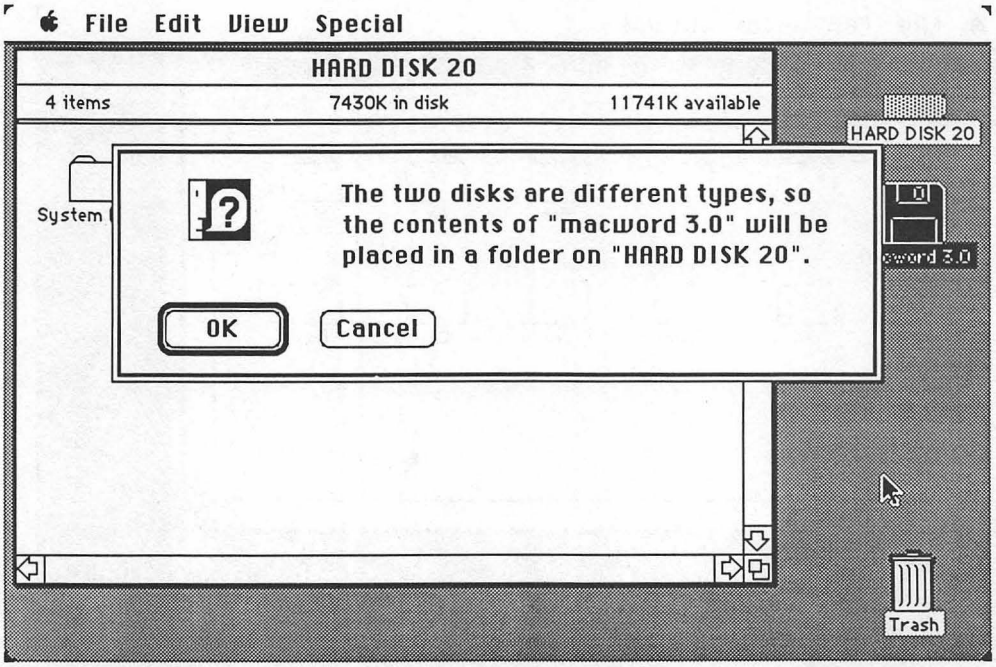


Figure 4-11. Different disk type alert box.

your best bet is to open the floppy icon. Then choose Select All from the Edit menu. Now drag the selected icons into the folder of your choice.

Hidden Files

Some applications have hidden files. If you copy an application with hidden files to the hard disk file by file, the necessary hidden files will not be copied. To avoid this problem, drag the entire disk icon to the hard disk icon. This way everything including the hidden files will be copied. Then move the System, Finder, printer files, Scrapbook, and

Clipboard files to the trash. You already have a copy in the System folder on your hard disk and Finder gets confused if there is more than one System.

Troubleshooting

You may get your disk organized exactly the way you want it, begin to work, and then discover that an application that worked perfectly well when you ran it from a floppy will not run from your hard disk. A few applications are copy protected in a way that won't let you run them from a hard disk. Fortunately we're seeing fewer and fewer of these. The trend is away from copy protection at best or at worst requiring a floppy to be inserted for verification at startup.

You might have an application that is not copy protected, but still will not run from inside a folder on the hard disk. A few applications, if they haven't been updated, will not run with the HFS. If your application does not work properly inside a folder, try moving it and its documents to the root directory of the hard disk and run it from there. In most cases this will work. If it's an application you use often, it's probably worth your time and money to contact the manufacturer for an HFS upgrade. Running it from the root directory will work, but doesn't take advantage of the HFS's folder nesting.

There may be an occasional application that will not run even if it is in the root directory. In that case you have no choice but to run it from a floppy. If you remove the System folder from the floppy disk, the application will use the System from the hard disk and you'll at least get a slight increase in operating speed that way.

Set Startup

Don't forget about the Set Startup command on the Special menu. It's a nice convenience. If you have an application you want to automatically

start each time you turn on your Mac, click on it to select it and choose Set Startup from the Special menu. If you change your mind later and want to start with the Finder just as you did before, simply open the System folder, click on Finder, and choose Set Startup again. When you restart, you'll be back to the desktop.

CHANGING THE PLAN

If after you work with your hard disk for several weeks, you determine you really have an inadequate organization plan, all is not lost. Because the HFS lets you move files around between folders, reorganizing is really quite simple. You simply create new folders and nest them according to your revised plan. Then move the contents of the old folders into the new ones, and throw away the folders you no longer need. As long as you've reasonably identified your files and folders, that is, given them descriptive names, you'll have little trouble moving them to a new folder.

CHAPTER 5

Guidelines for Basic Use

STARTING AN APPLICATION

Typically when you worked with existing documents and applications on floppy disks, you started the application and opened the document in one step by clicking on the document icon. This is one of the advantages of the Mac — it "knows" which document uses which application, and all you need to know is which document you want to work on. This is more like working with pen and paper — you pick up the letter you want to work on and begin — and radically different from the IBM PC world where you need to know both how to start the application and how to load your letter or spreadsheet. This advantage carries over into the hard disk world and is even more dramatic. With floppy disks you had to have your application and documents on the same disk (or disks if you had an external floppy drive). Either the application

or the documents could be in folders, but the folders could not be nested. With a hard disk and HFS, you don't have to worry about which documents and which applications are on which disk — everything will be on the hard disk. The application can be buried in folders many levels deep, and the document can be nested in an entirely different set of folders, but the Finder will still be able to locate the application when you click on a document icon.

So if you have a hard disk organization scheme that stores applications in a separate folder than documents for that application, it's most efficient to start work on an existing document by double-clicking on the document icon. That way Finder does the work of locating the application, starts it, and loads the document, and when you save your changes or additions, Finder proposes saving the revised document in the current folder, that is the folder in which the document was originally located. So if you want to file it back in the same folder you took it out of, all you need do is accept the proposed response. You don't first need to work your way through a series of nested folders to find the correct one and then click the Save button. And even if you move a document or folder around to a new folder, Finder continues to track its location.

OPENING AND SAVING

Even though clicking a document icon is a shortcut to starting an application and loading a document, you may occasionally need to work your way through nested folders to find an application and then again to open or save a document. You might, for example, be starting a new document or starting your applications with SetStartup or with Switcher. The SetStartup command and Switcher only start applications; they don't load documents. (More information on using Switcher can be found in chapter 6.) With the HFS, locating an application inside nested folders, starting it, and then working through the

hierarchy to open or save a document in a different folder is very straightforward.

Here's an example of how it might work. Say you're starting a new Word document, a letter to your assistant about reimbursing you for expenses from your last trip. First you need to work through the nested folders to locate your Word application. To keep a neat desktop you've nested the Applications folder inside your Systems folder, so you first need to double-click the Systems folder and then the Applications folder. Now that you've located Word, double-click the application icon to start it, and type your memo.

You'll, of course, want to print a copy for your assistant, and then file your electronic original in the proper folder, in this case nested several levels deep inside a financial folder (see Figure 5-1).

Choose the Save As . . . command. Note that names of folders in the dialog box are dark and enabled while files are light gray and disabled. This is because you use this box to choose the folder you want to save your application in. Pull down the Folder menu just above the standard file dialog box, and choose HD 20 or your root directory (see Figure 5-2).

Now work your way through the folders down to the February folder by double-clicking Financial, then Taxes, then Expenses, then Travel, and finally February. Type a name for this expense memo and click the Save button (see Figure 5-3).

If you need to do any work on the memo, you can make changes and then choose Save or Save As . . . Finder will propose putting it in the folder you chose the first time you used Save As . . . , so you only have to locate the correct folder once. And if you quit Word, work on something else, and then need to open the memo again to add information you've just recently received, double-click on your memo from inside the February folder. Finder will take care of locating the Word application

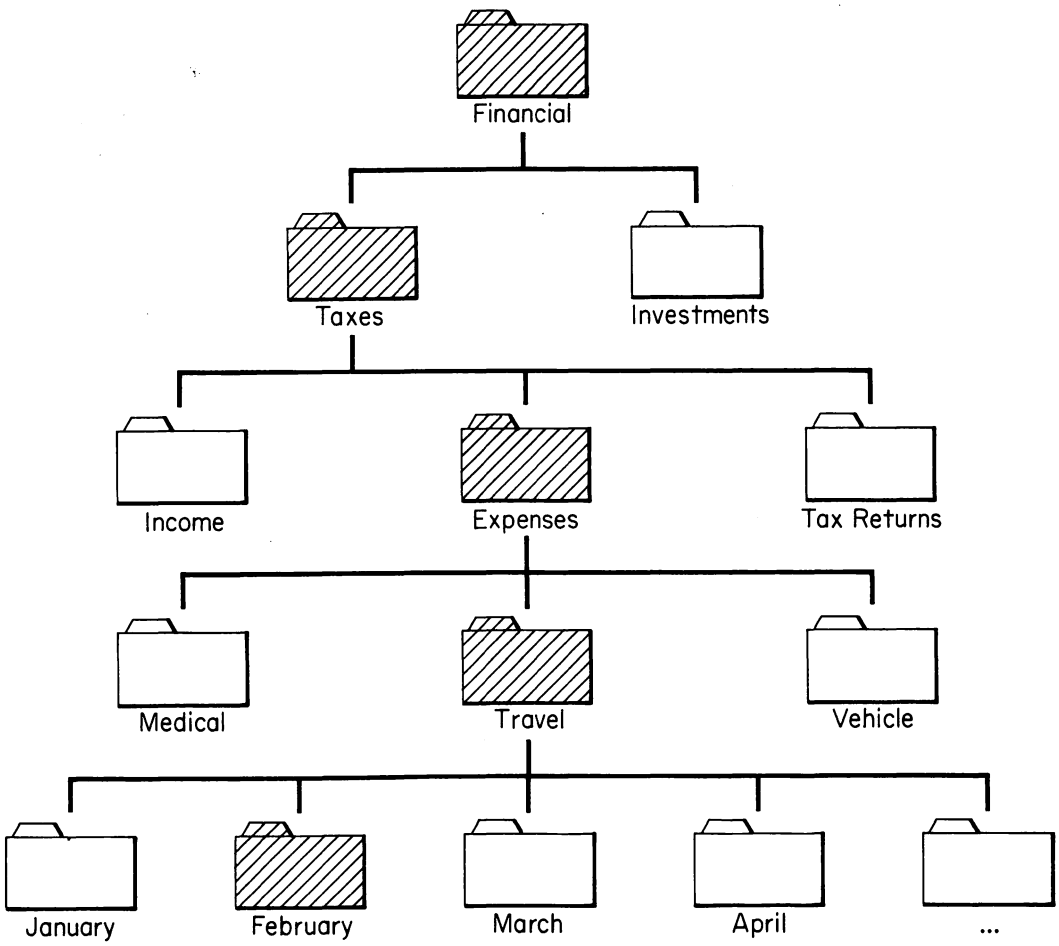


Figure 5-1. Financial folder nesting.

inside the Applications folder and when you save, Word proposes saving it in the current folder, February, which is just what you want.

When you finish with this memo and have saved the changes, you may

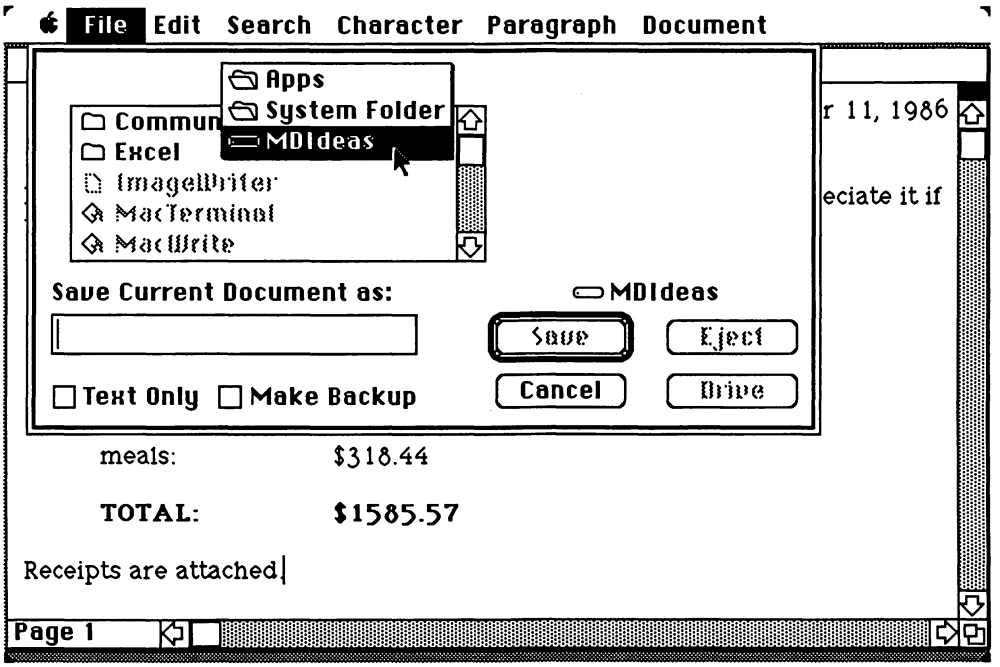


Figure 5-2. Save As dialog box with root directory selected.

wish to continue using Word, this time to work on an existing personnel memo. To do this first choose the Open command. Your personnel memo is inside a Personnel folder which is in the root directory of the desktop. So pull down the Folder menu above the dialog box, choose the root directory, and then open the Personnel folder (see Figure 5-4). Continue working your way through folders until you locate the memo you need.

It is possible to navigate through the folders in Open and Save dialog boxes without taking your hands from the keyboard. "Real typists" love keyboard equivalents to the mouse, and if your hands are already

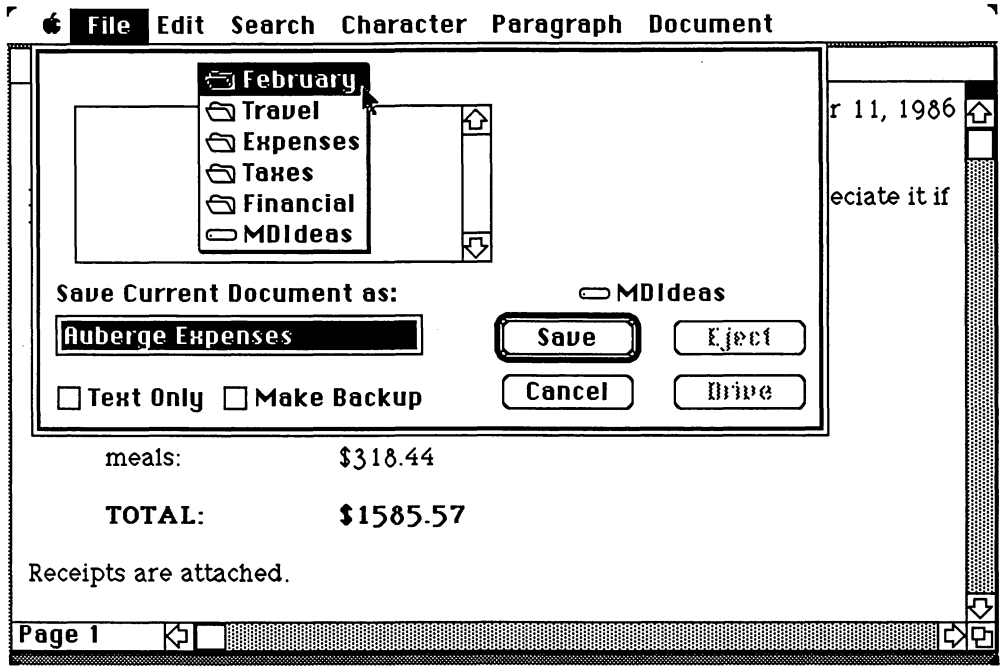


Figure 5-3. Memo in nested February folder.

on the keyboard, you may find them quicker than reaching for the mouse. Press the Command key and the up or down arrows to move through the folder hierarchy. When you locate the folder you want, press Return. Then use the up and down arrows without the Command key to select a file and press Return to carry out the command to open or save.

WORKING EFFICIENTLY

Each time you quit an application the Finder rebuilds the desktop to look the way it was when you left it. If you have a lot of files in the root directory, rebuilding the desktop can be terminally slow.

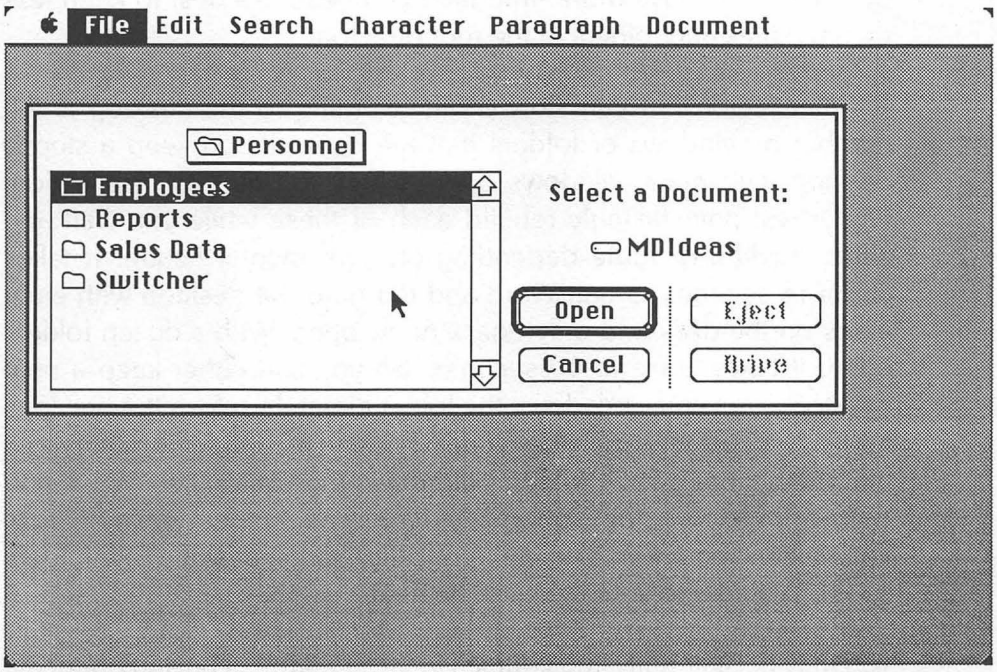


Figure 5-4. Personnel folder displaying nested folder.

Here are some statistics on time to rebuild the desktop based on number of files in the root directory. In this test only the Root folder was open.

Files in Root	Time to Rebuild Desktop
10	3 seconds
60	10 seconds
110	16 seconds
160	21 seconds
250	56 seconds
400	3.5 minutes

So unless you have more time than you need, it's best to keep less than ten files and folders in the root directory.

Another factor that can slow down returning to the desktop is the number of windows or folders that are open. If you keep a sloppy desktop with many windows open, when you quit an application, Finder will painstakingly rebuild each of these while you wait and either twiddle or fume depending on your mental health. It takes about 18 seconds to quit Word and return to the desktop with eight items on the desk and only one window open. With a dozen folders open, it takes close to 40 seconds. So you can either keep a neat desktop — as you work down the folder hierarchy, close the previous folder — or you can trick the Finder. Tricking the Finder is much more fun. As you open each folder, hold down the Option key. When you open a folder with the Option key depressed, Finder doesn't notice. Then when you return from an application to the desktop, the folders will be closed and your desk will be neat.

You may occasionally find yourself in the situation of having opened a dozen or more folders conventionally (without the Option key) and wishing for the old Close All command, a command available on early Finders that instantly tidied the desktop by closing all open folders. Don't despair — it is possible to simulate Close All like this: be sure nothing on the desktop is selected, then hold down the Option key and choose the Close command from the File menu. Magically, just like Close All, in one step all folders are closed.

COPY PROTECTION

What you'd like to do when you get a hard disk is copy everything in and relegate the old floppies to the back of a bottom drawer. Unfortunately, many programs use a copy protection scheme that makes this impossible. When you start the program from the hard disk, it requires you to insert the original floppy for verification. This means you still

have to keep your stack of old floppies handy to use each time you start an application.

Fortunately, there are ways around the problem but only after you read this disclaimer — these suggestions are only intended for people who have actually purchased the original software. Several programs are available for installing copy protected software on a hard disk so that it won't require use of a key disk to start. Two programs, HardDiskUtil ($89.95, FWB Software, 2040 Polk Street, Suite 215, San Francisco, CA 94109) and Copy II Mac ($39.95, Central Point Software, 9700 SW Capitol Hwy, #100, Portland, OR 97219) install almost any protected program onto your hard disk. Doing this requires patch files for different protection schemes and these files are supplied with the program. Neither of these programs is copy protected and each offers updates for new programs and protection schemes regularly. Copy II Mac offers regular upgrades to registered users at a reduced price, and HardDiskUtil offers updates on a San Francisco bulletin board open to all registered HardDiskUtil users for $20.00 per year.

REGULAR MAINTENANCE

One of the keys to using the space on a hard disk effectively sounds obvious, has been mentioned earlier, but is critical — name your folders and files carefully and descriptively. That way when you want to find a particular file, you will know exactly where to look, and you won't waste valuable time searching unnecessarily through folders.

Every week or two rebuild the desktop file, the one Finder uses to keep track of applications and documents, by holding down the Option and Command keys when starting up. What happens is that as you delete things from the disk and make changes, Finder does not physically

delete from the desktop file; for example, it keeps file information for applications and documents you've deleted. When you start with Option and Command, you'll see a dialog box like the one in Figure 5-5.

If you click YES, Finder cleans out unnecessary icons and other information, and this keeps the desktop file as small as possible. Since the desktop file is the one the HFS is constantly updating, you'll save time in the long run. Only rebuild the desktop on HFS disks and drawers. If you do it with MFS drawers, you'll lose your folders.

Regularly — at least every three to four months — houseclean your

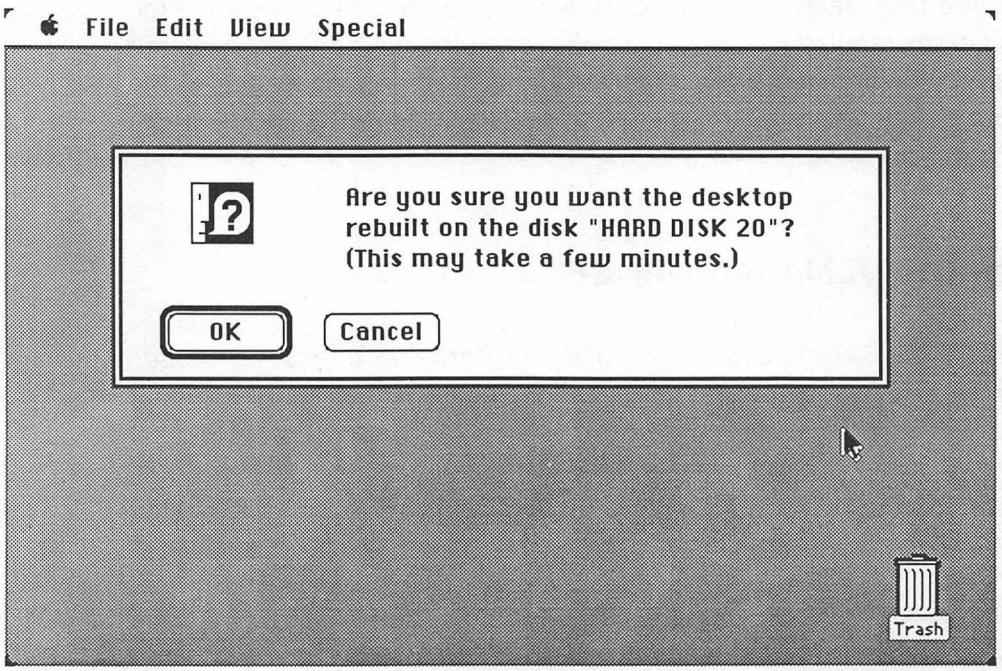


Figure 5-5. Confirmation dialog box.

disk. Go through each folder and delete files you no longer need. If there are some you're no longer using but you're reluctant to throw them into the trash, consider copying them to an archive floppy disk and storing them there. If, for example, you do a newsletter or monthly report, every six months consider moving old issues or reports to a floppy. This frees up space on the hard disk and makes it easier to find current projects. When I come to the end of a major project, after a book or article I've written is published, for example, I reluctantly move the entire folder to a floppy and delete the chapters, notes, and illustrations from the hard disk.

When you first get your hard disk, 20 megabytes seems like an almost infinite amount of space, and periodic housecleaning just makes working more efficient. But after several months or a year, the temptation to become a pack rat most likely will surface and you may find you've used almost all of the space on your disk. At this point periodic cleaning becomes a necessity.

CHAPTER 6

Tips For Optimal Use

This chapter contains tips for advanced users. Once you've mastered the basics, you may want to work even faster. If you find yourself waiting too long for the Finder to return when you quit an application, try the MiniFinder. If you find yourself in the middle of one task, need to switch to another task, and resent first saving and quitting the first application before you start the second, then Switcher is for you. If you've noticed a speed improvement when you added your hard disk but recently have watched its little light blinking indicating it's reading from the disk far too many times, perhaps setting a RAM cache will alleviate the problem. If you've had your disk for over a year and it just doesn't seem as fast as it used to, you may have a serious case of fragmented files; a solution follows. And if you're reaching the end of your megabytes, a public domain shareware program called PackIt can help.

MINIFINDER

The MiniFinder is an alternative to the Finder. It holds up to 12 documents or applications but no folders. It can't handle folders because it was added to Finder 4.1 before the HFS. It's ideal for working with more than one application, perhaps combining information from several applications, when you want to eliminate the annoying wait for the Finder to reconstruct the desktop when you quit one application and start another. Instead of reconstructing the desktop, the MiniFinder presents an abbreviated version of the desktop displaying only the applications and documents you've installed (see Figure 6-1).

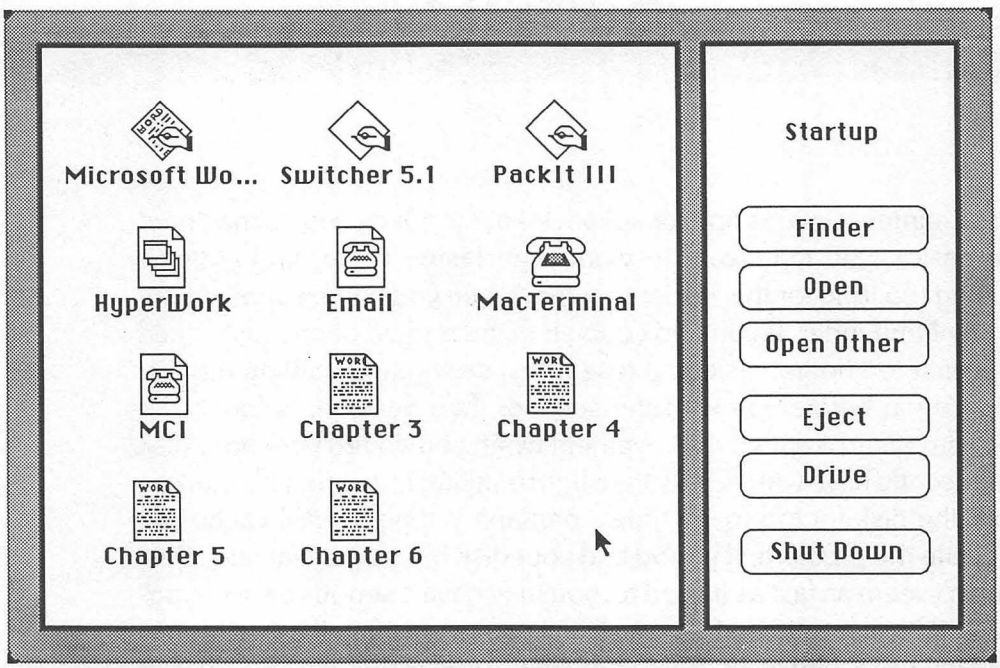


Figure 6-1. Sample MiniFinder.

The MiniFinder does have a back door. If you're working with it and suddenly discover you forgot to include an application or document you now need, note the Open Other button and the Finder button. Using them you can get to whatever you need.

Installing the MiniFinder

Basically first you choose which applications and documents you want to include in the MiniFinder, and then choose the Use MiniFinder command from the Special menu. But there is a catch, actually two catches. First, all applications and documents you select must be in the same directory window, either in one folder or loose in the root directory. Second, even though you can select applications, documents, and folders and when you choose Use MiniFinder you won't get an error, folders will not be included in the MiniFinder's display — it doesn't recognize them. So it's a good idea to drag the applications and documents you want to include from whatever folders you've nested them into a MiniFinder folder.

1. Decide which applications and documents you want to use in the MiniFinder. The MiniFinder can hold up to 12 items.
2. Put the applications and documents you want to use in a folder, perhaps named with the name of the task you'll be working on such as Newsletter and the initials MF for MiniFinder.

   If you find the MiniFinder useful and have plenty of disk space, you may wish to keep several permanent MiniFinder folders with applications and documents for tasks you do frequently. That way you won't have to set up a special folder each time you work, and refile your documents and applications when you finish. You could keep a duplicate copy of the applications in your MiniFinder folder, or you could simply store the applications there instead of in an Application or Task folder. This won't affect your work when you're no longer using the MiniFinder; the Finder will be able to locate the application no matter which folder it is in.

3. Use the Select All command on the Edit menu to select the applications and documents in your MiniFinder folder.
4. Choose Use MiniFinder from the Special menu. You'll see a dialog box like the one in Figure 6-2.
5. Click Install.

When the MiniFinder is installed, a MiniFinder icon is put in the System folder.

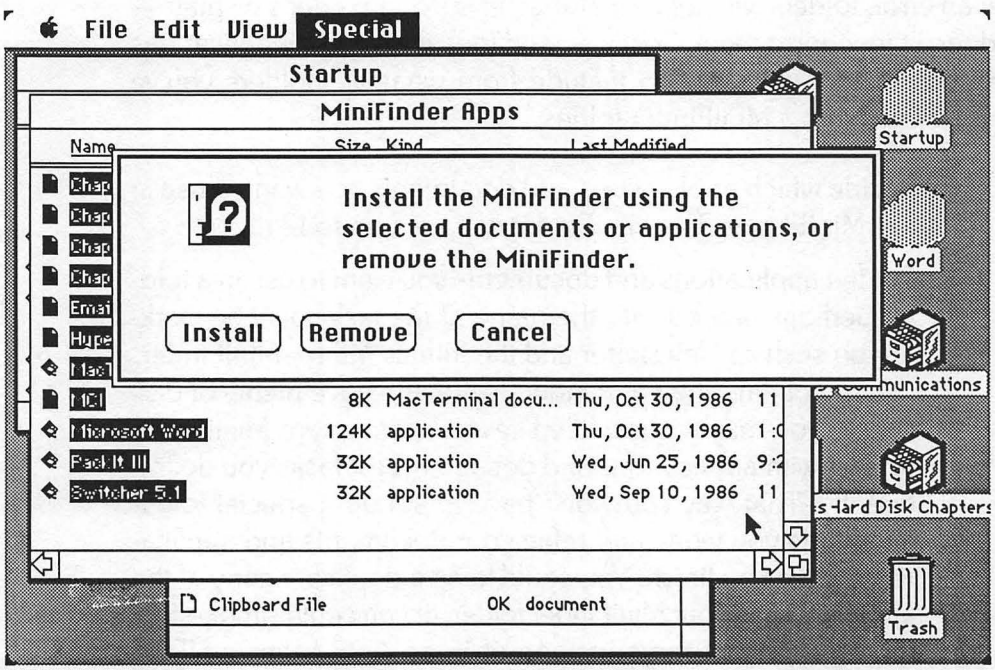


Figure 6-2. MiniFinder dialog box.

To Use the MiniFinder

1. To start the MiniFinder, simply open the System folder and double-click the MiniFolder icon. The Finder will then be replaced by the MiniFinder.
2. If you need an application not included in the MiniFinder, click the Open Other button. You'll see a dialog box similar to the Open dialog box for applications (see Figure 6-3).

   Note that this dialog box only lists applications, not applications and documents. Choose the application you need. Then use that application's Open command to get your document.

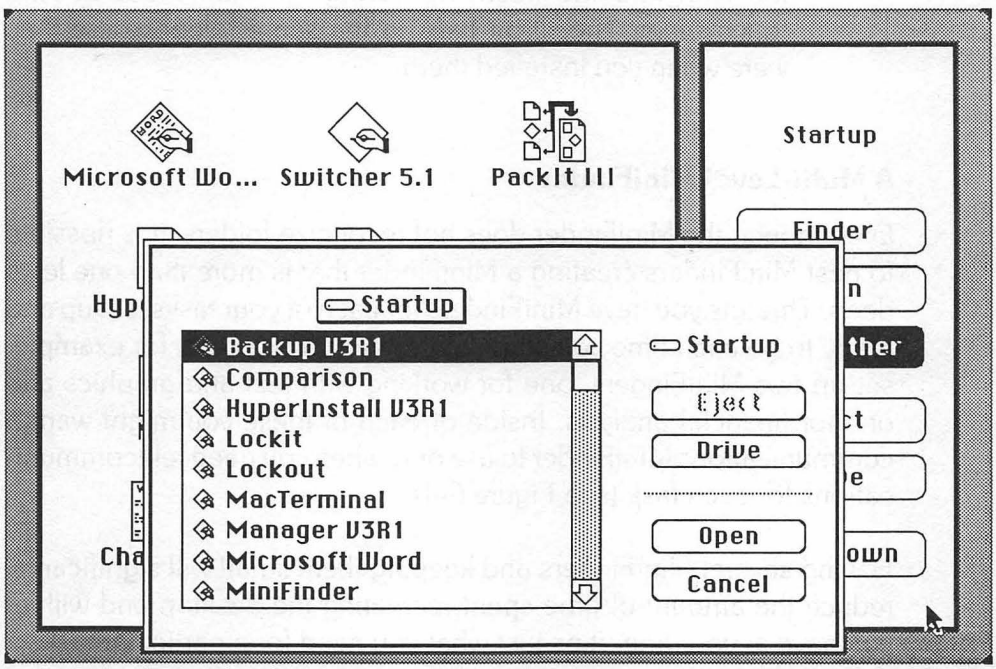


Figure 6-3. MiniFinder Open dialog box.

3. You can quit from the MiniFinder and shut off your system by clicking the Shut Down button and turning off your system as soon as the screen darkens. But unless you remove the MiniFinder before you quit, each time you start your system, you'll see the MiniFinder rather than the Finder.

To Remove the MiniFinder

1. Click the Finder button on the MiniFinder desktop.
2. Choose Use Minifinder on the Special menu and click the Remove button, or drag the MiniFinder icon into the trash.

   Don't worry about the applications and documents that were in the MiniFinder when you chose Remove or dragged the icon into the trash. Removing the MiniFinder doesn't touch them. They'll be back in the same folder where they were when you installed them.

A Multi-Level MiniFinder

Even though the MiniFinder does not recognize folders, it is possible to nest MiniFinders creating a MiniFinder that is more than one level deep. This lets you have MiniFinders for each of your tasks, set up and ready to go each time you start your system. You could, for example, set up two MiniFinders, one for working with text and graphics and one for financial analysis. Inside of each of these you might want a communications MiniFinder to use only when you need telecommunications for each task (see Figure 6-4).

Having several MiniFinders and keeping them small will significantly reduce the amount of time spent recreating the desktop and will let you have at your fingertips just what you need for a particular task.

But using multiple-level MiniFinders does require you to plan ahead

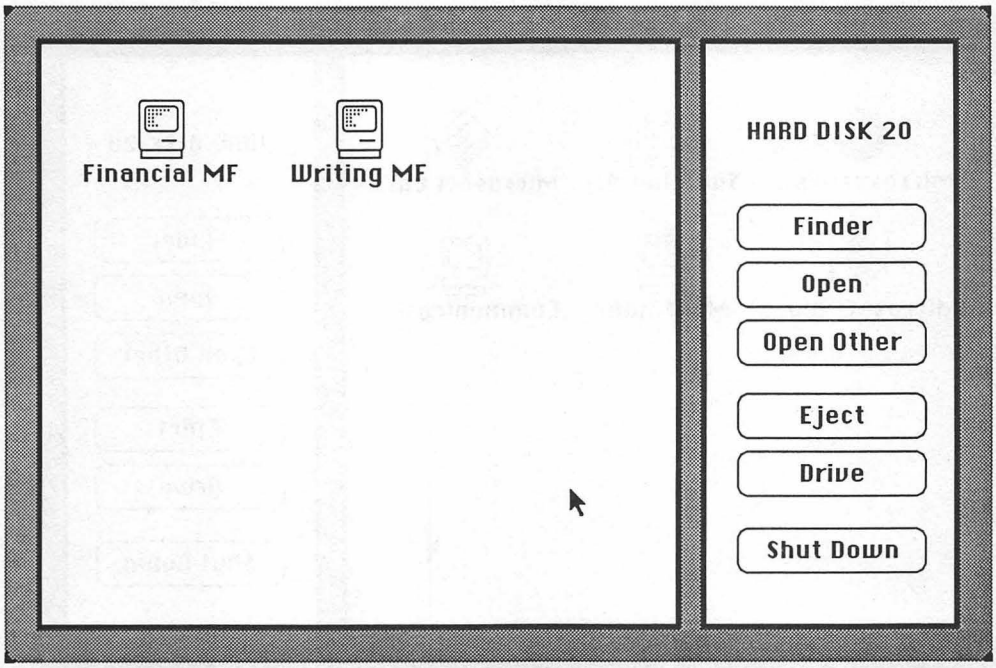


Figure 6-4a. First-level MiniFinder.

and set them up just the way you want them. It also requires a fair amount of disk space — if you use the same application for different tasks, you'll need a copy in each MiniFinder folder.

If you think multiple-level MiniFinder will work for you, here's how to create a two-level MiniFinder. You start with the secod level:

1. Get a folder and drag all the applications and documents you want in one of the second-level MiniFinders.
2. Select any one item on the desktop and choose Use Mini-Finder from the Special menu.

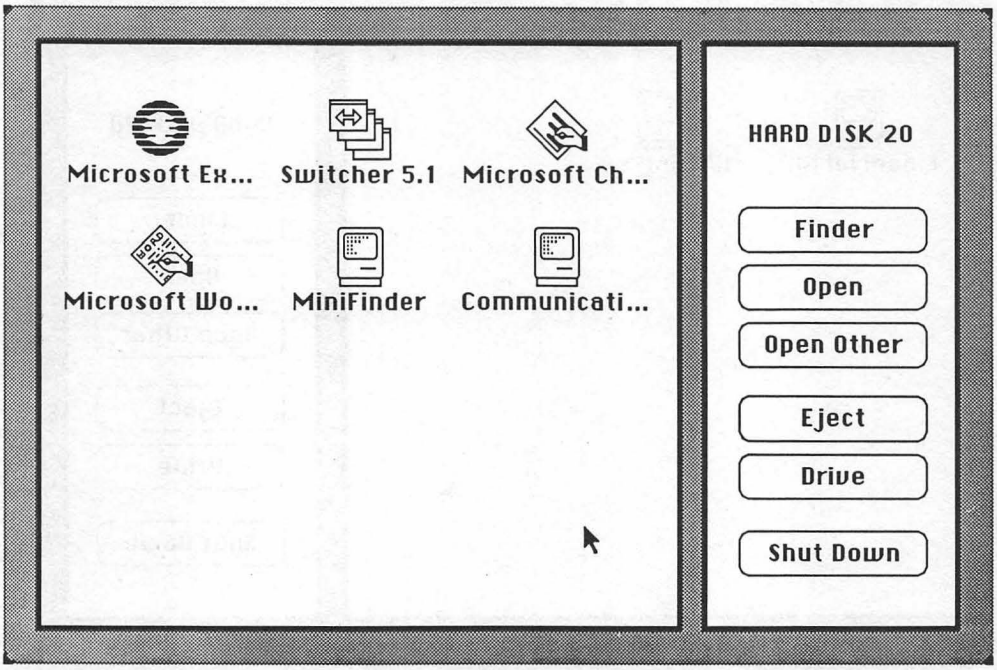


Figure 6-4b. Second-level MiniFinder.

This puts a Minifinder icon in the System folder.

3. Open the System folder and drag the MiniFinder icon into the folder with your second-level MiniFinder applications and documents.

   Your MiniFinder icon is your ticket out of that level of the MiniFinder to the next higher level.

4. Select everything in this folder — the applications and documents and the MiniFinder icon and once again choose Use MiniFinder from the Special menu.

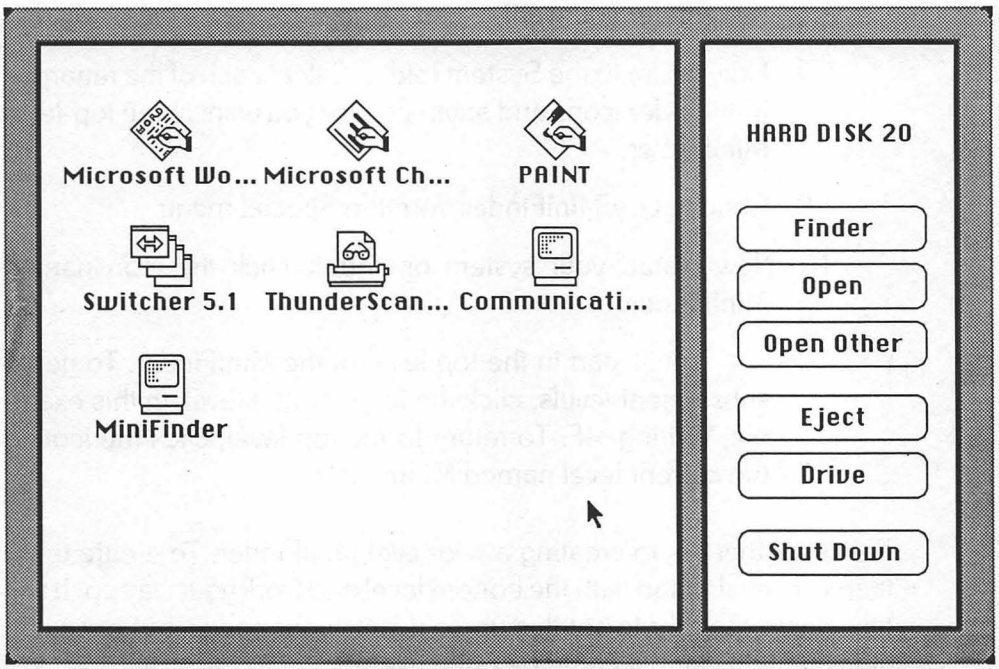


Figure 6-4c. Third-level MiniFinder.

This puts a new MiniFinder icon in the System folder. You need to rename this and it will be your entry from the top-level to the next lower-level MiniFinder. But the Finder won't let you do this directly. So what you must do is duplicate the MiniFinder icon and rename the copy.

5. Open the System folder, select MiniFinder, and choose Duplicate from the File menu.
6. Type a new name for copy of MiniFinder, something like Writing MF.

7. Follow steps one through six for each additional second-level MiniFinder you want.
8. Now return to the System folder. Select each of the renamed MiniFinder icons and anything else you want in the top-level MiniFinder.
9. Choose Use MiniFinder from the Special menu.
10. Now restart your system or double-click the icon named MiniFinder.

    You'll start in the top level of the MiniFinder. To get to subsequent levels, click the icon for that level, in this example, Writing MF. To return to the top level, click the icon in the current level named MiniFinder.

That's all there is to creating a two-level MiniFinder. To create more than two levels, start with the bottom level and work your way up. It will take some thought to set this up so it suits your needs, but once you get what you want, it could be quite useful.

SWITCHER

Switcher is another option you may find useful depending on the type of work you do. If you're frequently interrupted, have to quit one application and start another, or if you frequently exchange information between applications, Switcher may let your work more quickly. What you do is set up Switcher with the applications you plan to use. Switcher puts a little double-pointed arrow in the upper right corner of the screen. Then when you want to switch from one application to the next, you simply click the arrow. You don't need to save and quit one application before starting another. And when you return to the first application, you'll be exactly where you were when you clicked the arrow.

I use Switcher all the time. Even if I'm basically using one application such as Microsoft Word, I often install the Finder as a second application. That way any time I can look at the Finder, do what I need to do — look at other directories or floppy disks, and then instantly get back to where I was when left Word. Sometimes I install MacTerminal for my electronic mail account as well. Then if I'm working in Word and feel the urge to check my Email, it's not a major time commitment. I can click the arrow and MacTerminal with my Email file loaded is ready to go. I can check my mail and return to Word without wasting time quitting, starting, quitting and restarting.

Switcher allows you to install up to four applications if you have a 512K Mac and eight with 1024K. But what you can actually install depends on the amount of memory you have and the types of applications you install. With 512K typically I can only install one large application such as Microsoft Word and one other application, like MacTerminal or Finder. With a Mac Plus I can usually install whatever I need. There is one interesting quirk I've found with Switcher. If you install Finder as one of your apps, then switch to Finder and start another application, that application replaces Finder. When you quit that app, Finder is no longer installed in Switcher; you need to install it again.

Here's how to set up Switcher:

1. Double-click on the Switcher icon. You'll see a screen like the one in Figure 6-5.
2. Double-click on the icon in the top row. You'll see a standard Open dialog box like the one in Figure 6-6.
3. Work your way through the directories until you find the application you want to install. Select it and click Open.

   Now you'll see the double-headed arrow in the upper right corner of the screen (see Figure 6-6).

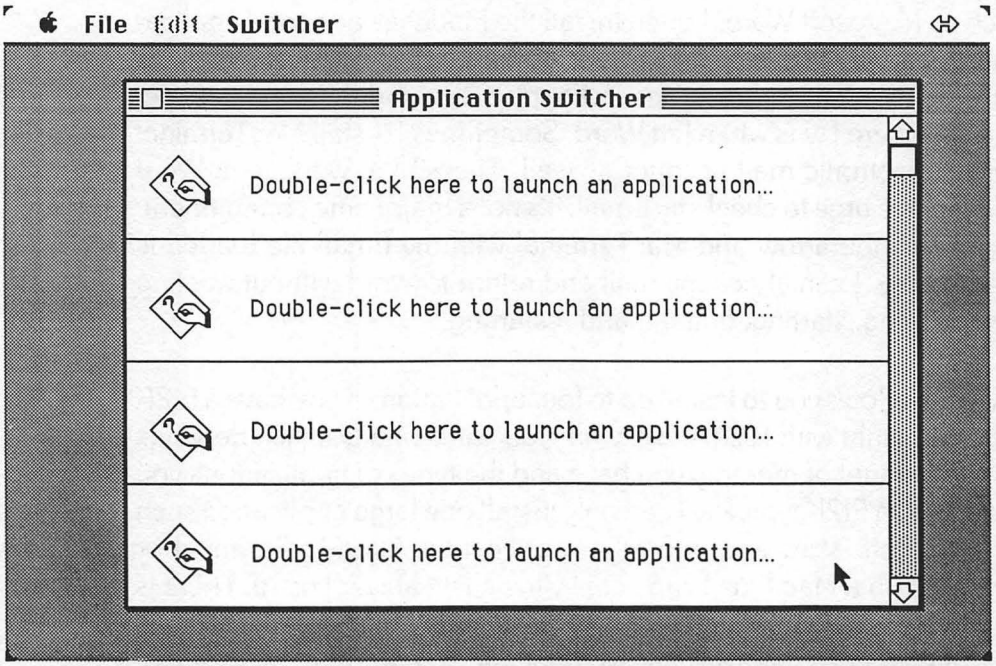


Figure 6-5. Switcher install screen.

Clicking the right or left point of the arrow takes you from one application to the other. Clicking the middle of the arrow takes you back to the Switcher install screen.

Note that when you install an application, Switcher replaces the generic icon with the icon for the application you've installed and also displays the amount of memory it's allocated for that application. If you want to allocate a different amount of memory for your application, first quit the application. Then return to Switcher and choose Configure then

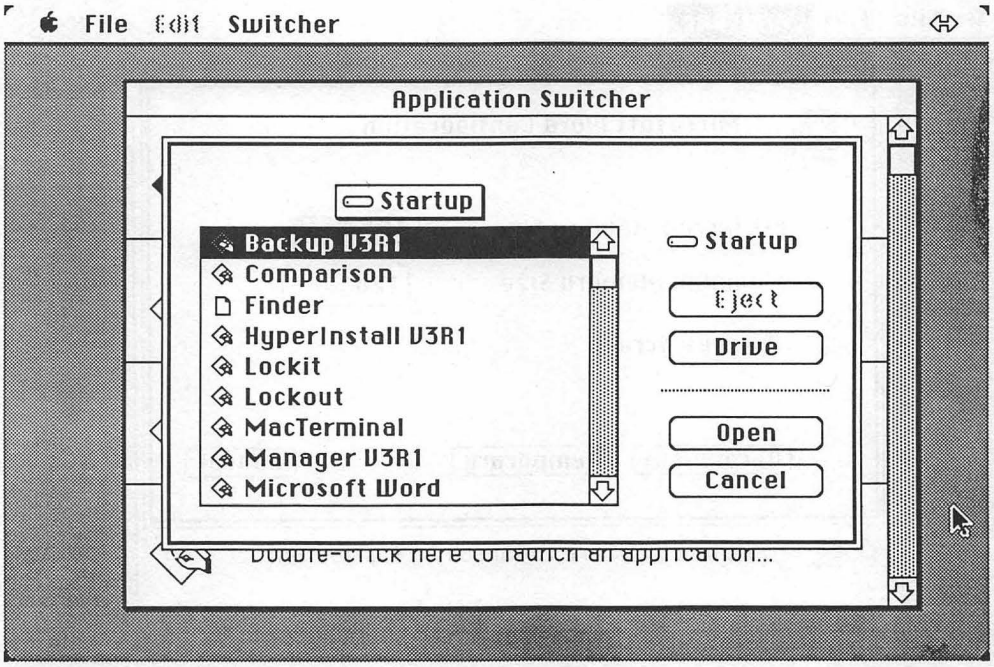


Figure 6-6. Switcher Open dialog box.

Install . . . from the Switcher menu. You'll see a dialog box like the one in Figure 6-7.

Specify the amount of memory you need and click Permanent. That way Switcher will remember this figure and each time you install the application will allocate that amount of memory.

4. Continue installing as many applications as you need or have room for.
5. Choose the Save Set . . . command from the File menu. Name your file and click Save.

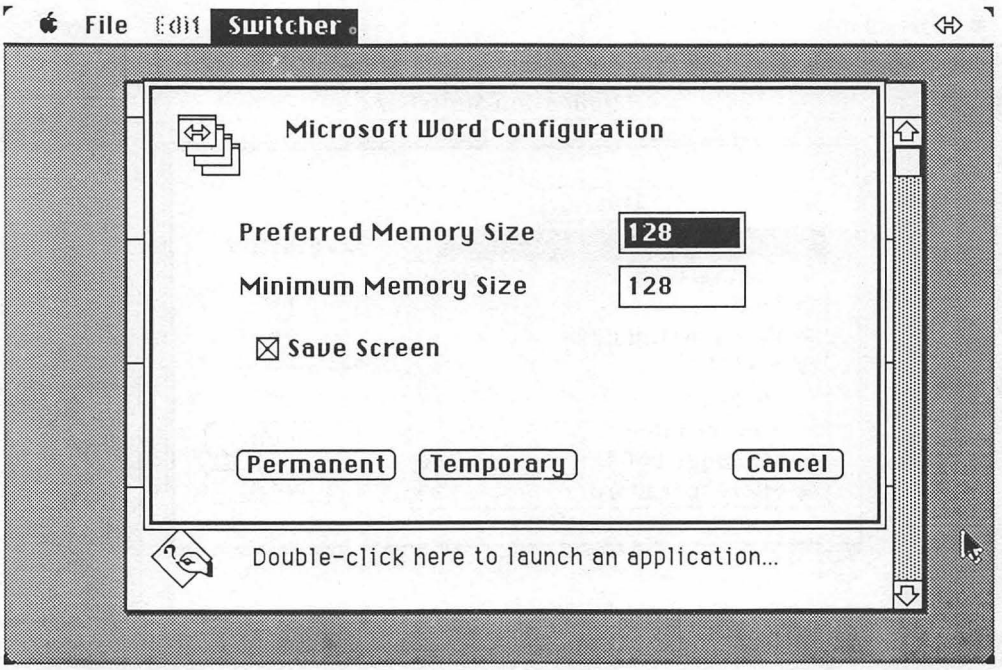


Figure 6-7. Switcher Config dialog box.

What this does is create a Switcher document for these applications. The next time you want to use exactly the same group of applications, instead of installing them again, simply double-click on your Switcher document and Switcher will take care of installing them for you.

When you want to leave Switcher, first quit any active applications. Then choose Quit from Switcher's File menu.

Switcher, written by Andy Hertzfeld, is available from Apple dealers for $19.95.

THE RAM CACHE

The RAM cache is an area of memory set aside to store information from the disk. Here's how it works. The first time an application reads from the disk, that information is stored in the cache. The next time the application makes a request for information from the disk, it first checks the cache; if the information is there, it skips the disk read and gets the information from RAM which is much faster. The cache also stores information you write to the disk. If, for example, you're editing a document in MacWrite, and make several changes, rather than writing these to the disk, they are stored in the cache, and only written out to the disk when the cache is full or when you save or quit. Reading from and writing to memory rather than disk can really speed up work. When the cache is full, blocks are copied back to disk in the least-recently-used order.

You turn the cache on or off and allocate the amount of memory from the Control Panel (see Figure 6-8). Click on the arrows to raise or lower the amount of cache. It is allocated in 32K segments. The amount of cache you use or whether you use any at all depends on the type of work you do. A little cache — 32K — is almost always worth it. Typically a small cache can hold volume and folder information that will speed up opening and saving dialog boxes. If you work with large applications like Word and Excel, a large cache — 128 to 160K — is a good idea. That way it can hold both a large document or worksheet and segments of code and improve performance considerably. On the other hand, if you're a programmer, you may wish to turn off the cache. You may want all the memory for the compiler and your development. If you're working with an especially volatile application, you may not want to risk losing the information in the cache if the system crashes.

There are no hard and fast rules for the amount of memory to allocate to the cache. If you allocate too little, there will be too much turnover, and the disk and cache will be constantly thrashing and caching, causing performance to decrease. You'll be able to hear the disk

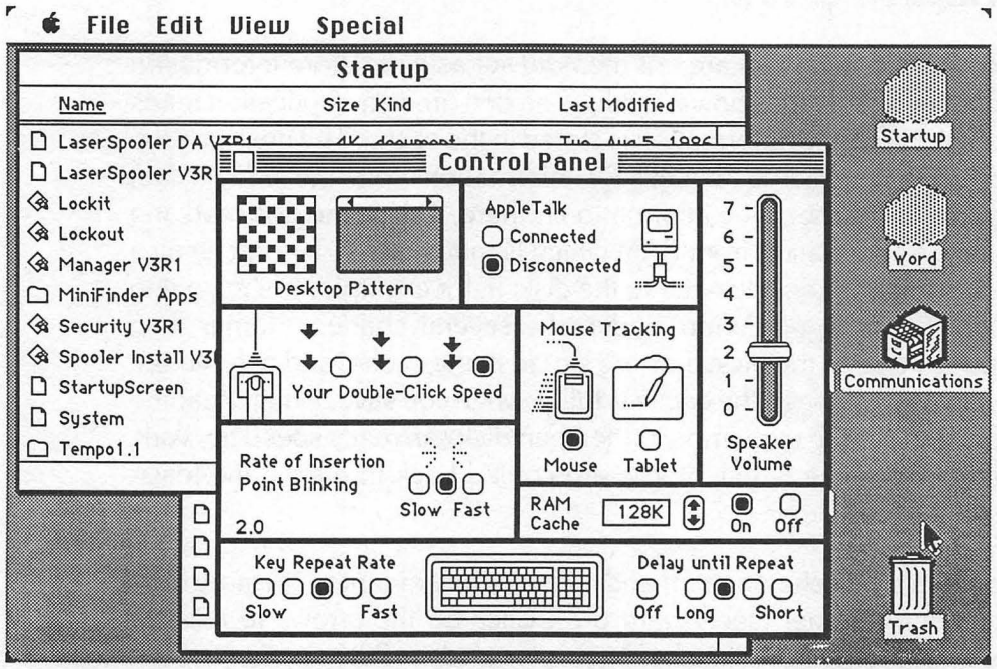


Figure 6-8. Control Panel — RAM cache selected.

reading and writing and depending on the type of hard disk, watch the light flicker on and off. If this happens frequently, increase the amount of cache, and try again. The trial and error system is the best way to figure out what you need.

Keep in mind when you vary the amount of cache that the settings in the Control Panel do not go into effect until the next time you quit or open an application. This is because the cache has to be allocated before the application takes the memory it needs.

Some hard disks such as HyperDrive 2000 provide their own RAM

cache. General Computer recommends that if you use their RAM cache (set in the Manager), you turn off Apple's. Or if you use Apple's, you disable theirs.

FRAGMENTED FILES

After many files have been created and erased on a disk, they tend to fragment. They wind up in pieces scattered around on the disk and require more head movements to read. This fragmentation happens gradually and eventually can significantly slow down performance.

There is a solution, but it's not quick and easy. The solution is to copy everything on the disk to floppies. If your hard disk is full, you'll need 25 800K floppies. Then initialize your disk, and copy everything back. Now your files will again be in contiguous blocks.

Chapter 8 contains a backup program that can make this process less painful. It's not something you need to do every week or even every month. Once or twice a year should be sufficient.

PACKIT III

PackIt III is a handy program you can use to combine several files and compress them so they take up less space on your disk. If you're close to the top of your 20 MB's, have cleaned out everything you don't "need", and aren't quite ready to invest in an additional 20 MB's, you might consider PackIt. I used PackIt on four 20K files and it compressed the group of four to 48K — That's 40% less than before packing!

PackIt can best be used to compress files you most likely won't need to use again, perhaps ones you're archiving. Although PackIt also provides a utility for unpacking files, it's not something you'll want to do to files you use regularly.

A message you see on the screen when you start says, "PackIt is shareware. It isn't free." It works like most shareware. You can get a copy from a friend, and if you like it, you send the author a fee, in this case, $10.00. Or you can order the latest version directly from the author, Harry Chesley, 1850 Union Street, #360, San Francisco, CA 94123, for $20.00.

Here's how you compress files:

1. Put the files you which to compress and the PackIt program into a folder.
2. Select all the files you wish to compress and then double-click on the PackIt icon.

   You'll see a dialog box that looks like the one in Figure 6-9.
3. Click the Compression box and click OK.

PackIt then asks you to give the compressed file a name.. It proposes the name packit.pit, but you can change this. I suggest leaving the .pit extension because this identifies it as a compressed PackIt file.

As PackIt works, it displays a graph that shows you exactly what it's doing. When it is finished you'll have the compressed file, and your original files will be intact; it doesn't replace your original files with the compressed files.

To Unpack Files

Unpacking is simple. Simply select the packed file and double-click. PackIt will go to work, display a screen similar to the one you saw when you packed a file, and suggest the original names for the files as it unpacks them one by one.

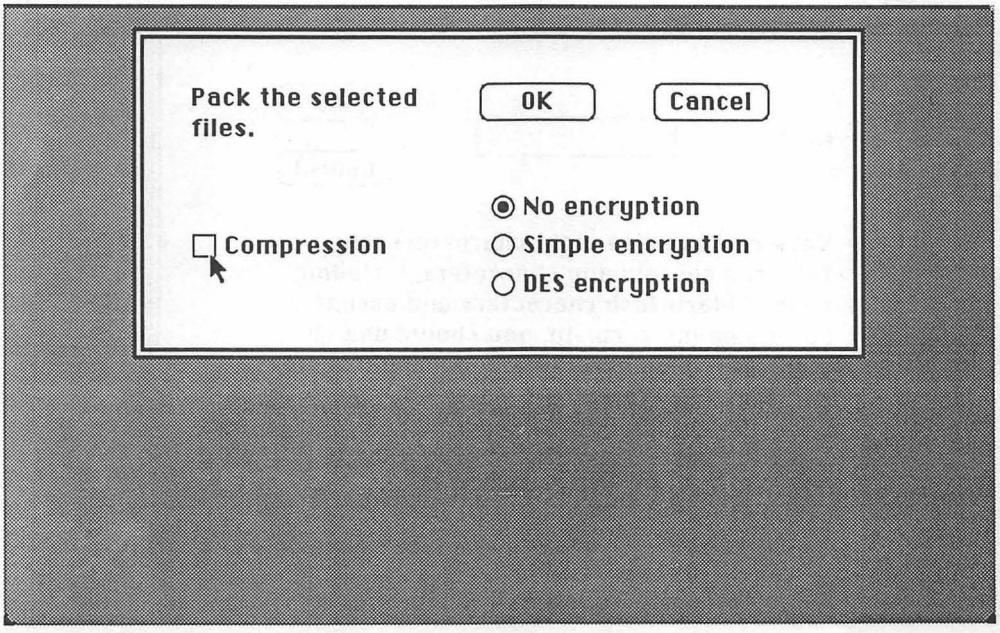


Figure 6-9. PackIt dialog box.

To Encrypt Files

As you pack files, PackIt III also has the capability to encrypt them either with standard or DES (Document Encryption Specification) encryption. If you click the standard or DES encryption button, you'll see a dialog box asking for an encryption key (see Figure 6-10).

This is the word you must specify before PackIt will unpack files. So in addition to using this to save disk space, you can also use it to password protect sensitive information. People who want to open a compressed and encrypted PackIt file need both the PackIt program and the encryption key or password. When you try to unpack an encrypted

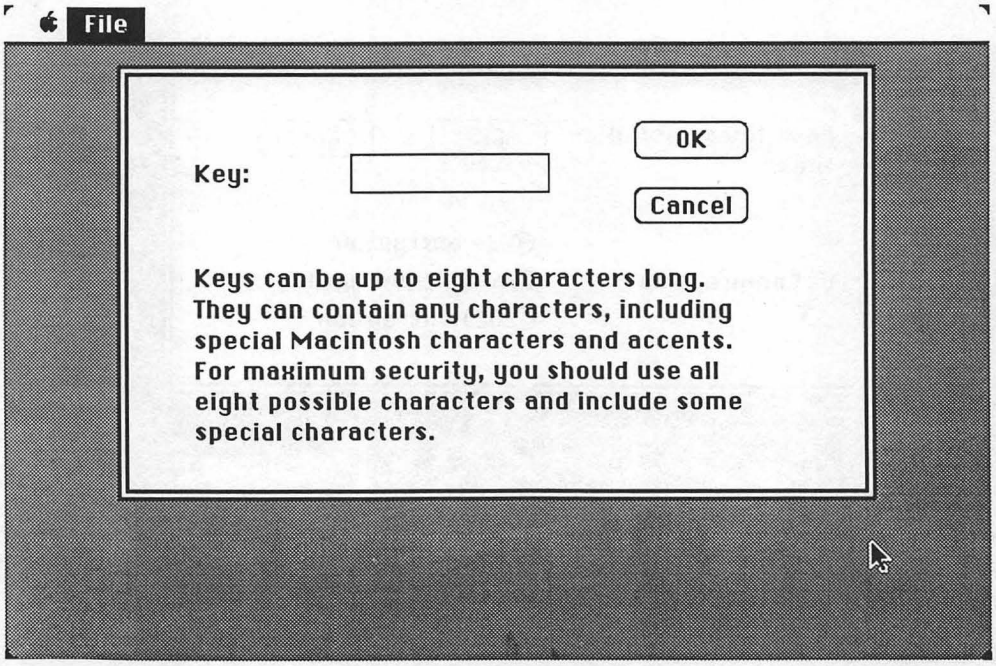


Figure 6-10. Encryption key dialog box.

file, you'll see a similar dialog box asking for the key. Only after you supply it will PackIt go to work, unpack the file, and make it usable. For more information on hard disk security, see chapter 9.

Other utilities to optimize hard disk performance are needed. Some hard disks come with a good set of utilities. Others such as the Apple HD 20 do not. More and more shareware hard disk utilities are becoming available. And later chapters include three necessary utilities — one for locating lost files, one for backing up your hard disk, and one for protecting your entire hard disk using a password.

CHAPTER 7

Security

PUT A LOCK ON IT

When you used only floppies, if you had sensitive information, you could simply lock the disks in your desk drawer. Now, with all your information on your hard disk, how can you keep an unauthorized person from starting your system and snooping around? And what if someone alters sensitive data on your disk? How will you know? Better yet, how can you prevent it?

There are several approaches to the hard disk security problem depending on whether one or more than one person uses the disk. First, if several users share the same computer and hard disk, the best solution is a disk like HyperDrive that comes with its own security software. It lets you protect individual volumes or drawers with a password. This way each user can keep sensitive information in his or her own secure drawer. Second, if several users share a hard disk via a

network, the network file server software should allow for protected directories. That way you can put sensitive information in a directory that only you can use. Chapters 10 and 11 have more detailed information on networks.

Encrypt or Password Protect

If you're an individual user with a hard disk without security software, such as the Apple HD 20, you basically have two choices, encrypting selected files or password protecting the entire disk. You can use an encryption program such as PackIt III discussed in chapter 5. When you finish working with a file, you run it through an encryption program and only someone with the password or key will be able to decrypt it and use it. This is perhaps the safest of all solutions, but is is time-consuming, particularly if you must encrypt most of your files. Another solution is to password protect the entire disk. When you start your computer, you need to enter a password to get to the Finder. If you don't know the password, you can't use the system. This is the easiest solution and can keep casual prowlers from sensitive information.

LOCKIT

This chapter presents a password program called LOCKIT. Once LOCKIT is installed, when you start your system, instead of the Finder, you'll see a screen like the one in Figure 7-1.

As you type in the password, it won't be displayed in the dialog box. Instead you'll see X's representing each character you type. This is so someone walking by will not be able to glance over your shoulder and read your password. When you finish typing your password and press Return, the password dialog box will be replaced with the Finder.

If you type the wrong password or type your password incorrectly, you'll see the message shown in Figure 7-2.

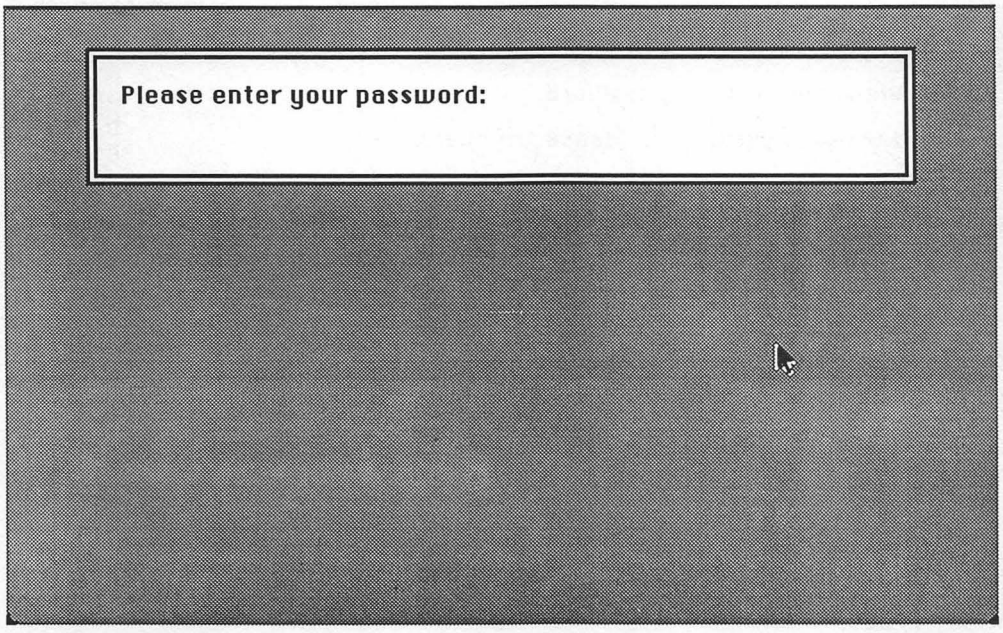


Figure 7-1. Initial LOCKIT screen.

Just type the password again and when you finally get it right, you'll get to the Finder. Initially the password is set to "PassWord". The program is case-sensitive, so typing "password" or "Password" will not do. You must type the exact combination of uppercase and lowercase letters.

You can, of course, change the password so yours is unique. To change the password, type the current password. When you finish, instead of pressing Return, press Command Return. This displays the change password dialog box with the current password selected (see Figure 7-3). Simply type a new password, it will replace the old password in the text box, and click the OK button.

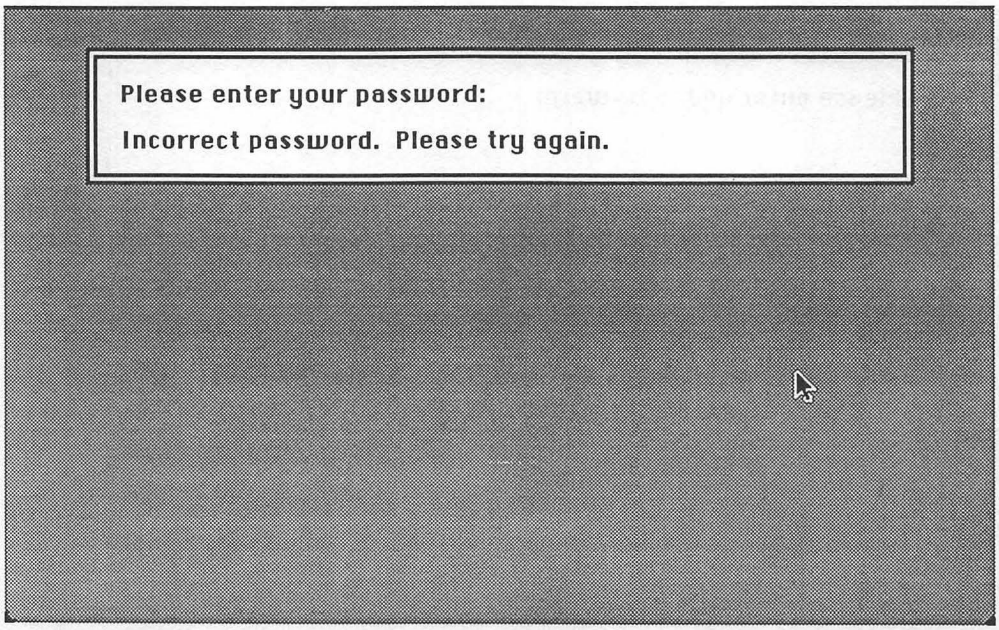


Figure 7-2. Incorrect password dialog box.

The way you install LOCKIT is to place it in the root directory or System folder. Click on it to select it and choose Set Startup from the Special menu. Now restart your system and instead of starting with the Finder, it will start with LOCKIT.

LOCKIT has some limitations; it is not foolproof. It's major limitation is this. It only protects your hard disk if you boot from the hard disk. This is certainly the easiest and fastest way to start the system, but you could bypass LOCKIT if you booted from a floppy. If the floppy didn't also contain the LOCKIT program installed with the Set Startup command, the system would start right up without asking for a

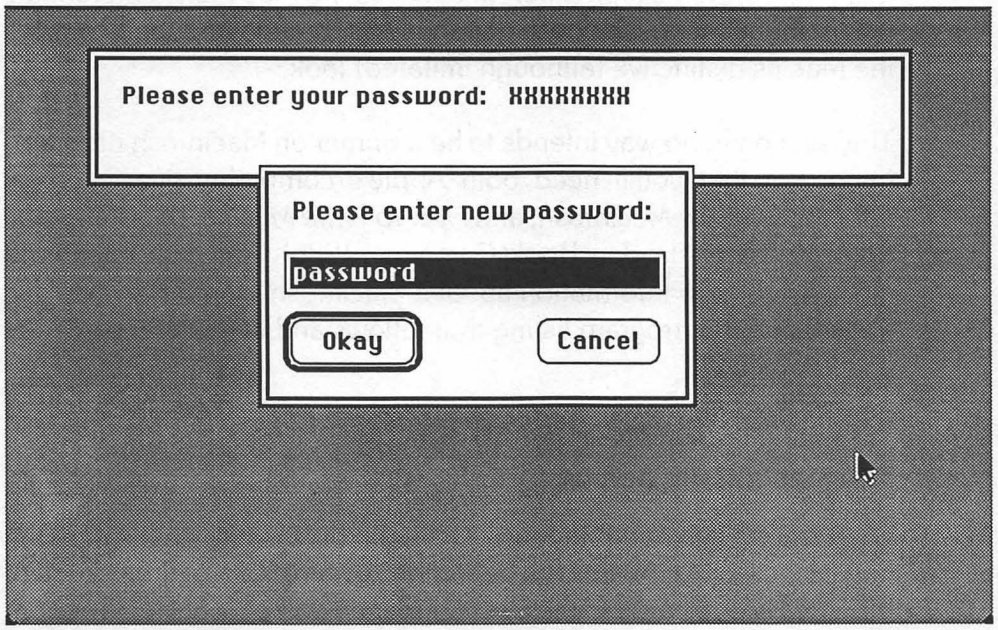


Figure 7-3. Change password dialog box.

password. So as a precaution, add LOCKIT to all bootable floppy disks you have lying around your system. This will deter casual browsers and prowlers. It won't, however, deter a determined hacker; very little will deter a determined hacker.

A WORD ABOUT MACINTOSH PROGRAMMING

If you've programmed on other computers but not on the Mac and glanced ahead at the program listing for LOCKIT, you may have been surprised to notice that it looks quite different than the same program

would look if it were written for a conventional computer like the IBM PC. Mac programs look different because the Mac looks different. Its user interface with windows, dialog boxes, menus and the like gives the Mac its distinctive (although imitated) look.

This section in no way intends to be a primer on Macintosh programming. For that you'll need both Apple's comprehensive technical reference, *Inside Macintosh*, and *How to Write Macintosh Software* by Scott Knaster (Hayden Book Company, 1986). Rather it intends to give you enough information about a Macintosh program so you can understand the program listing that follows and, if you choose, enter and compile it.

The Macintosh Toolbox

What makes programming the Macintosh different is what's in ROM. Part of the ROM contains the operating system, but a major portion of the ROM is dedicated to the User Interface Toolbox. This toolbox consists of hundreds of callable routines to create the windows and dialog boxes and menus — the visual parts of Macintosh programs. What this means to a programmer is that instead of writing code for all the standard functions, you call toolbox routines to do the work. So the challenge is not painstakingly writing code to draw each box and button, but finding the right routine to call to do the job for you. A friend who helped write the LOCKIT code said, "The Mac isn't hard to program — it's just hard to figure out how to program."

Macintosh programs have a unique feature called "callbacks". A callback works something like this. If your program has a dialog box defined as a resource with a text box similar to the LOCKIT program, you attach your own procedure, for example, to the press of the Return key. Then you give the operating system dialog box or event handler the addresses of this procedure. That's all you have to worry about. You don't have to monitor keystrokes and mouse moves, filter

out the ones you don't need, and take the appropriate action when you get what you want. When the Return key is pressed, the operating system "calls back" your routine and that part of your program gets control. So the job of the programmer is to define actions specific to his or her own program, use the toolbox functions to hook it into the operating system, and then let the operating system and toolbox handle things common to all applications.

As you'll see with the LOCKIT program, there is some overhead to setting up the visual interface for a Macintosh program. That's why writing a very small Macintosh program may take more lines of code than writing an equivalent very small program for the IBM PC. But because routines for the visual parts of the program are in the toolbox and can be called over and over, as you add features to a Macintosh program and it grows, it will grow more slowly than a PC program will when you add equivalent features.

MPW — The Macintosh Programmer's Workshop

The LOCKIT program is written in MPW, Macintosh Programmer's Workshop, Pascal. The body of the program is pretty much standard Macintosh Pascal. You can use it with most Pascal compilers. The resource section of the program is unique to MPW Pascal. To oversimplify a bit, there are two types of resource compilers — MPW's and everyone else's. For that reason we've also included a resource list for TML Pascal as an example that should work with most compilers other than MPW.

LOCKIT — THE PROGRAM

This section explains the LOCKIT program procedure by procedure. If you want the LOCKIT program but are not a programmer and have no intention of becoming one, your best bet is to skip the rest of this chapter

and simply use LOCKIT. If, on the other hand, you want to learn something about Mac programming and perhaps enter and compile this program, read on.

Overview

The main LOCKIT program starts by doing the standard initialization and then sets up the enter password dialog box. Next it uses the built-in ROM call, ModalDialog, to activate and manage the dialog box. It gives ModalDialog a pointer to function CheckPassword and returns control to the operating system. When the operating system has an event for our program, it wakes up our program and begins the CheckPassword function (see Figure 7-4).

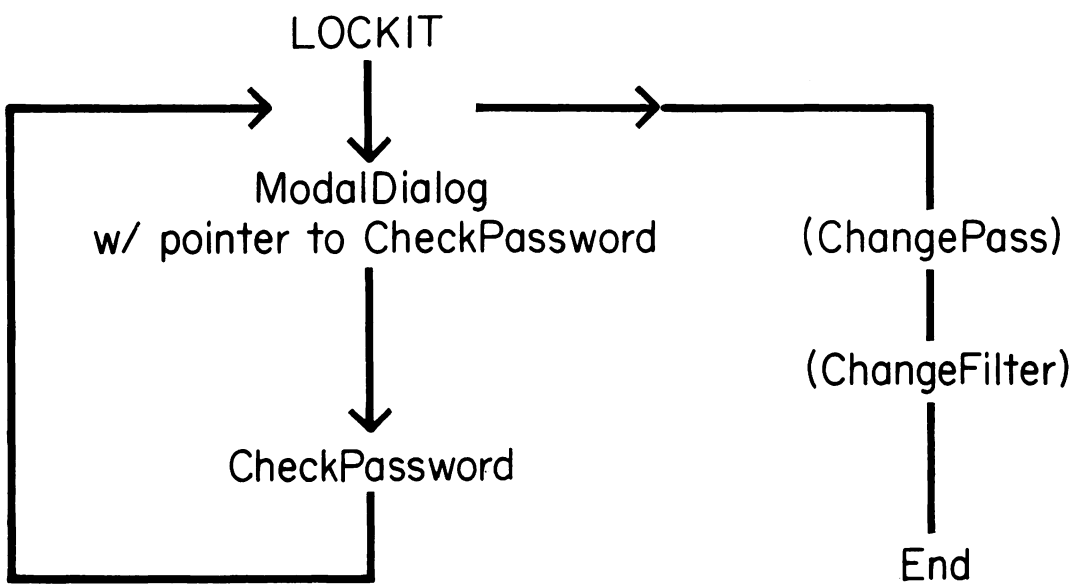


Figure 7-4. LOCKIT diagram.

CheckPassword does the major work in the LOCKIT program. It collects keystrokes for the user string and compares this to the password. If the user string is not the password, it displays an incorrect password message and waits for another string. CheckPassword also monitors whether the user pressed Return or Command-Return and then returns to the main program.

If Command-Return was pressed at the end of the password, then the main program calls procedure ChangePass. This sets up the change password dialog box and uses ModalDialog with the address of ChangeFilter. ChangeFilter takes a press of the Return key and makes it the equivalent of clicking the OK button in the change password dialog box, then returns to ChangePass. It takes the new password from the dialog box, disposes of the dialog box and returns to the main program. All the main program has left to do is get rid of the check password dialog box and end.

Now we'll look at the listing procedure by procedure starting at the beginning:

Figure 7-5a.

[LOCKIT.LIST]

```
{   Lockit - A sample application written in MPW Pascal  }
{       ©1987 - Bantam Books }
{       This program asks for a password that must be entered before }
{       allowing the user into Finder. The password can be  }
{       changed by pressing 'Command Return' at the end of the }
{       password line.                                  }
```

```
{ To make Lockit from MPW

    Pascal -s Lockit.P
    Rez -o Lockit.R.O Types.R Lockit.R
    Duplicate -y Lockit.R.O Lockit
    Link Lockit.P.O ∂
        MPW:PLibraries:Paslib.O ∂
        MPW:Libraries:Runtime.O ∂
        -oc "????" -ot "APPL" -o Lockit
}

PROGRAM Lockit;

    USES
        {$LOAD pas.s}
        Memtypes,
        Quickdraw,
        OSIntf,
        ToolIntf,
        PackIntf;

    CONST
        PWShowItem = 2;
        BadPWItem = 3;

        OkayBtn = 1;
        CancelBtn = 2;
        PWText = 3;
        ButtonFrame = 5;

        RtnKey = 13;

        PassID = 100;
        BadPWStr = 200;

    VAR
        checkDialog : DialogPtr;
        passWord, showWord : Str255;
        itemHit : INTEGER;

{------------------------------ CheckPassword ------------------------------}

    FUNCTION CheckPassword (theDialog: DialogPtr; VAR theEvent: EventRecord;
                            VAR itemHit: INTEGER): BOOLEAN;
```

```
{ The dialog filter procedure for reading the password.
  Displays the typed password with 'X's and
  returns 1 if the password was entered correctly,
  2 if the user pressed 'Command Return' }

VAR
    itemType : INTEGER;
    itemHdl  : Handle;
    itemBox  : Rect;
    theChar  : CHAR;

BEGIN
    CheckPassword := FALSE;
    { Is it a key event? }
    IF theEvent.what = keyDown THEN
    BEGIN
        theChar := CHR (BAND (theEvent.message, charCodeMask));
        { Was it the 'Return' key }
        IF theChar = CHR (RtnKey) THEN
        BEGIN
            { Did the password match? }
            IF passWord = GetString (PassID)^^ THEN
            BEGIN
                CheckPassword := TRUE;
                IF BitAnd (theEvent.modifiers, cmdKey) = 0 THEN
                    itemHit := 1 { They pressed 'Return' }
                ELSE
                    itemHit := 2; { They pressed 'Command Return' }
            END
            ELSE
            { They didn't type the right password }
            BEGIN
                passWord := '';
                showWord := '';
                GetDItem (checkDialog, BadPWItem,
                          itemType, itemHdl, itemBox);
                SetIText (itemHdl, GetString (BadPWStr)^^);
            END;
        END
        ELSE
        { They typed a character, add it to the password }
```

```
                BEGIN
                    passWord[0] := CHR (ORD (passWord[0])+1);
                    passWord[Length (passWord)] := theChar;
                    showWord := CONCAT (showWord, 'X');
                    { Remove the 'Bad Password' message. }
                    GetDItem (checkDialog, BadPWItem,
                              itemType, itemHdl, itemBox);
                    SetIText (itemHdl, '');
                END;
            { Re-Display the 'X's }
            GetDItem (checkDialog, PWShowItem,
                      itemType, itemHdl, itemBox);
            SetIText (itemHdl, showWord);
        END;
    END;

{------------------------------  ButtonUpdate  ------------------------------}

    PROCEDURE ButtonUpdate (theWindow: WindowPtr; theItem : INTEGER);

        { The update procedure used in ChangePass for the ring around the 'OK' Button }

        VAR
            itemType : INTEGER;
            itemHdl : HANDLE;
            itemBox : RECT;
            oldPen : PenState;

        BEGIN
            GetPenState (oldPen);
            PenSize (3, 3);
            GetDItem (theWindow, ButtonFrame, itemType, itemHdl, itemBox);
            FrameRoundRect (itemBox, 18, 18);
            SetPenState (oldPen);
        END;

{------------------------------  ChangeFilter  ------------------------------}

    FUNCTION ChangeFilter (theDialog: DialogPtr; VAR theEvent: EventRecord;
                           VAR itemHit: INTEGER): BOOLEAN;

        { The dialog filter procedure for changing the password.
          Changes the 'Return' key into the 'OK' button }
```

```
    VAR
        temp : BOOLEAN;
        theChar :  Byte;

    BEGIN
        ChangeFilter := FALSE;

        IF theEvent.what = keyDown THEN
        BEGIN
            { Click the 'OK' button if they pressed the Return key }
            IF BAND (theEvent.message, charCodeMask) = RtnKey THEN
            BEGIN
                ChangeFilter := TRUE;
                itemHit := OkayBtn;
            END;
        END;
    END;

{------------------------------  ChangePass  ------------------------------}

PROCEDURE ChangePass;

    { Allow the user to change the password }

    VAR
        changeDialog :  DialogPtr;
        itemType  : INTEGER;
        itemHdl     : Handle;
        itemBox     : Rect;
        strHdl      : StringHandle;

    BEGIN
        { Get the 'Change password' dialog. }
        changeDialog := GetNewDialog (200, NIL, POINTER (-1));

        { Connect the update routine for
            the ring around the 'OK' Button. }
        GetDItem  (changeDialog, ButtonFrame, itemType, itemHdl, itemBox);
        SetDItem  (changeDialog, ButtonFrame,
                   itemType, @ButtonUpdate, itemBox);

        { Set the edit text item to display the current password. }
        GetDItem  (changeDialog, PWText, itemType, itemHdl, itemBox);
```

```
        SetIText (itemHdl, passWord);
        { Select the entire password so that it can easily be re-entered. }
        SelIText (changeDialog, PWText, 0, 32767);

        { Call the dialog }
        ModalDialog (@ChangeFilter, itemHit);

        { Did they press the 'OK' Button? }
        IF itemHit = OkayBtn THEN
        BEGIN  {Insert new password into resource list}
            GetDItem (changeDialog, PWText, itemType, itemHdl, itemBox);
            GetIText (itemHdl, passWord);
            strHdl := GetString (PassID);
            SetString (strHdl, passWord);
            ChangedResource (Handle (strHdl));
        END;

        { Get rid of the dialog. }
        DisposDialog (changeDialog);
    END;

{------------------------------  Lockit  ------------------------------}

BEGIN
    { Initialize the system }
    InitGraf (@thePort);
    InitFonts;
    FlushEvents (everyEvent, 0);
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs (NIL);
    InitCursor;

    { Clear the password }
    passWord := '';
    { And the displayed word. }
    showWord := '';

    { Get the 'Enter password' dialog. }
    checkDialog := GetNewDialog (100, NIL, POINTER (-1));
```

```
    { Call the dialog. }
    ModalDialog (@CheckPassword, itemHit);

    { Did they press 'Command Return' at the end of the password? }
    IF itemHit = 2 THEN
        ChangePass;

    { Get rid of the dialog. }
    DisposDialog (checkDialog);
END.
```

Declarations

The USES section lists files from the ROM toolbox used in this program. If you haven't used these before, it's a good idea to print them out — they're written in standard Pascal — so you can see which functions are available.

All items in the CONST list except RtnKey refer to the resource list at the end of the program. A diagram of LOCKIT resources is shown in Figure 7-5b. The VARs are your standard global variables. passWord contains the string for the word the user enters. ShowWord contains the X's that are displayed as the user enters the password.

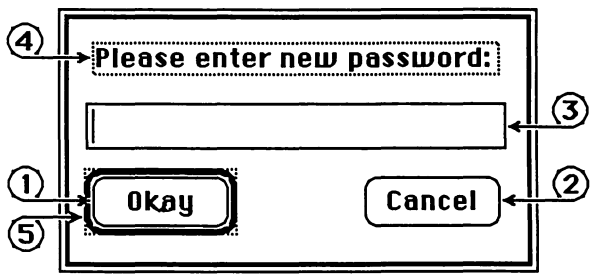
CheckPassword

CheckPassword is called by ModalDialog with information about an event that CheckPassword is interested in, in this case a keypress, and ModalDialog provides 32 bits of information about the event in the form of an event record. CheckPassword first looks at the message part of the event record. The line:

```
theChar: = CHR (BAND (theEvent.message, charCodeMask))
```

extracts the eight-bit character code for the keypress. If the Return key was pressed, then the user has finished entering the password and CheckPassword checks for a match. If the password matches, there is still one more check — to see if the user pressed Return only or pressed Command-Return to change the password. It uses a BitAnd function with the cmdKey bit. CheckPassword then sets the variable itemHit to 1 for Return only and 2 for Command-Return and returns to the main program.

Item 1: 'Please enter your password' StaticText.
Item 2: 'X's StaticText.
Item 3: 'Incorrect password. Please try again.' StaticText.



Item 1: 'Okay' Button.
Item 2: 'Cancel' Button.
Item 3: New password EditText.
Item 4: 'Please enter new password:' StaticText.
Item 5: The ring around the 'Okay' button (UserItem)

Figure 7-5b. LOCKIT resource diagram.

If the password did not match, CheckPassword has more work to do. It first blanks outs passWord and showWord, the X's, and displays the bad password string — "Incorrect password. Please try again."

The next section of CheckPassword handles keypresses that are not the Return key. It collects the characters that make up the password. The first line:

```
password[0]: = CHR (ORD (passWord(0)+1)
```

increments the length of the password. The first byte of a character string in MPW Pascal stores the current length of the string. Then it stores the string character by character. For each character, it adds an X to the variable showWord. And in case this is not the first try at the password, this section also contains two lines to remove the Incorrect Password . . . message. The last part of the function displays the X's each time a key other than the Return key is pressed.

ButtonUpdate

ButtonUpdate is a short procedure called by ChangePass to draw the ring around the OK button. This ring indicates that the OK button rather than the Cancel button is the default.

ChangeFilter

ChangeFilter is a short procedure needed for events you want interpreted in a non-standard way. The ChangePass dialog box expects you to click the OK button. ChangeFilter interprets pressing the Return key as the equivalent of clicking the OK button.

ChangePass

ChangePass, not surprisingly, is the procedure that lets you change your password. It sets up a change password dialog box and connects

to ButtonUpdate for the ring around the Ok button. Then it puts the current password in the text box and selects it so the user can simply type a new password without first dragging to select it. Then it, like the main program, calls ModalDialog, only this time it sends the address of ChangeFilter. The event handler then calls ChangeFilter which does its work and returns to ChangePass. If the user clicked OK rather than Cancel, ChangePass gets the new password from the text box and changes the resource that stores the password. Last, it throws away the change password dialog box.

LOCKIT

The main program does the initialization, clears out the password string and the showword string, the one that holds the X's. Then it sets up the password dialog box and passes control to ModalDialog with the address of CheckPassword, the procedure that gathers a password string and checks it against the "real" password. ModalDialog calls CheckPassword when it has an event such as a keypress that CheckPassword wants. After CheckPassword does its work, it returns to the main program. Then if the user pressed Command-Return to change the password, the main program calls ChangePass. After ChangePass is completed, the main program throws out the password dialog box and is finished.

RESOURCES

Figure 7-6 shows the resource list for LOCKIT in MPW Pascal. Figure 7-7 shows an equivalent list for TML Pascal. You should be able to use the TML format for any Pascal compiler other than MPW.

The resources in the resource listing are a text representation of graphic objects. You can enter them in text form like the listings as shown in Figures 7-6 and 7-7, and compile them just as you do Pascal code (see Figure 7-8).

Figure 7-6.

[MPWResource.Lockit]

```
/* The resource specification for the application 'Lockit' */
/* Presented in MPW resource compiler format */

/* The 'Enter password' Dialog. */
resource 'DLOG' (100, preload){
    {50, 52, 104, 462}, /* Window's BoundsRect  */
    dBoxProc,           /* Window's ProcID  */
    visible,
    noGoAway,
    0x0,                /* Window's RefCon (unused) */
    100,                /* Resource ID of DITL  */
    ""                  /* Dialog title  (unused)  */
};

/* The 'Change Password' Dialog. */
resource 'DLOG' (200, preload) {
    {110, 140, 220, 350}, /* Window's BoundsRect  */
    dBoxProc,             /* Window's ProcID  */
    visible,
    noGoAway,
    0x0,                /* Window's RefCon (unused) */
    200,                /* Window's RefCon (unused) */
    ""                  /* Dialog title  (unused)  */
};

resource 'DITL' (100, preload) {
    {  /* array DITLarray: 3 elements */
       /*  [1]  */
       {8, 8, 24, 200},
       StaticText {
           disabled,
           "Please enter your password:"
       };
       /* [2] 'X's shown as password is typed */
       {8, 208, 24, 408},
```

```
        StaticText {
            disabled,
            ""
        };
        /* [3] 'Incorrect password.' */
        {32, 8, 48, 272},
        StaticText {
            disabled,
            ""
        }
    }
};

resource 'DITL' (200, preload) {
    {  /* array DITLarray: 5 elements */
        /* [1] 'Okay' button. */
        {72, 8, 96, 72},
        Button {
            enabled,
            "Okay"
        };
        /* [2] 'Cancel' button. */
        {72, 136, 96, 200},
        Button {
            enabled,
            "Cancel"
        };
        /* [3] New password. */
        {40, 8, 56, 200},
        EditText {
            disabled,
            ""
        };
        /* [4] */
        {8, 8, 24, 200},
        StaticText {
            disabled,
            "Please enter new password:"
        };
        /* [5] Ring around 'Okay' button. */
        {68, 4, 100, 76},
        UserItem {
```

```
            disabled
        }
    }
};

/* The Current password. */
resource 'STR ' (100, preload) {
    "PassWord"
};

/* The 'Incorrect password' string. */
resource 'STR ' (200, preload) {
    "Incorrect password.  Please try again."
};
```

But because resources represent graphic objects, it's easier if you can see the resources you're creating on the screen as graphic objects, make them look just they way you want them to in your program, and then somehow turn this into source code. And that's what the resource editor is for. With it you can visually create menus, windows, and dialog boxes; size them, and move items around by dragging them. (You can even use it to change the look of commercial applications you use!) The resource editor comes with a decompiler that produces source code for the resources you've created. In addition, it also produces a text or human-readable file. That way if you ever lose your only copy of the source or object code, you can recreate it automatically by compiling the text file. If you use the resource editor and decompiler, the process look like Figure 7-9.

Take a look at the resource editor that comes with your compiler. It's an amazing tool and a great way to make icons for your program as well.

Now that you understand this program, it will be easy to enter and

Figure 7-7.

[TMLResource.Lockit]

```
* The resource specification for the application 'LockIt'.
* Presented in MDS RMaker format (used by TML Pascal).

Lockit.Rel

* The 'Enter password' Dialog.
Type DLOG
      ,100  (4)                    ;; Resource ID
Unused Title                  ;; Dialog title (unused)
50 52 104 462                 ;; Window's BoundsRect
Visible NoGoAway
1                             ;; Window's ProcID
0                             ;; Window's RefCon (unused)
100                           ;; Resource ID of DITL

* The 'Change password' Dialog.
Type DLOG
      ,200  (4)                    ;; Resource ID
Unused Title                  ;; Dialog title (unused)
110 140 220 350               ;; Window's BoundsRect
Visible NoGoAway
1                             ;; Window's ProcID
0                             ;; Window's RefCon (unused)
200                           ;; Resource ID of DITL

* The 'Enter password' Dialog's item list.
Type DITL
      ,100  (4)
3                             ;; 3 items

StaticText Disabled           ;; Item # 1
8 8 24 200
Please enter your password:
```

```
StaticText Disabled             ;; Item # 2 - 'X's shown  as password is typed
8 208 24 408

StaticText Disabled             ;; Item # 3 - 'Incorrect password.'
32 8 48 272

* The 'Change password' Dialog's item list.
Type DITL
      ,200  (4)
5                               ;; 5 items

Button Enabled                  ;; Item # 1 - 'Okay' button.
72 8 96 72
Okay

Button Enabled                  ;; Item # 2 - 'Cancel' button.
72 136 96 200
Cancel

EditText Disabled,              ;; Item # 3 - New password.
40 8 56 200

StaticText Disabled             ;; Item # 4
8 8 24 200
Please enter new password:

UserItem Disabled               ;; Item # 5 - Ring around 'Okay' button.
68 4 100 76

* The Current password.
Type STR                        ;; 'STR ' (Space is required.)
      ,100  (4)
PassWord

* The 'Incorrect password' string.
Type STR                        ;; 'STR ' (Space is required.)
      ,200  (4)
Incorrect password.  Please try again.
```

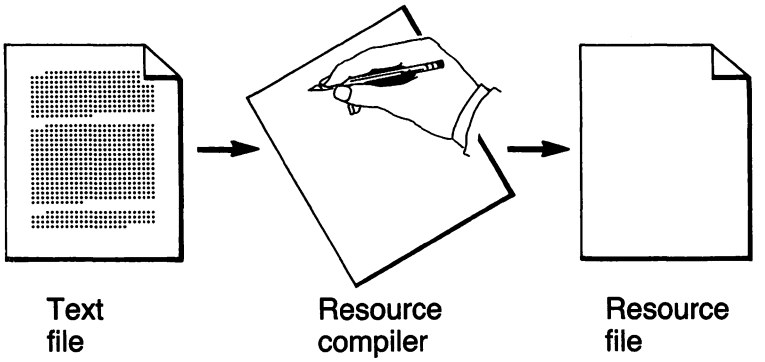


Figure 7-8. Resource compiler steps.

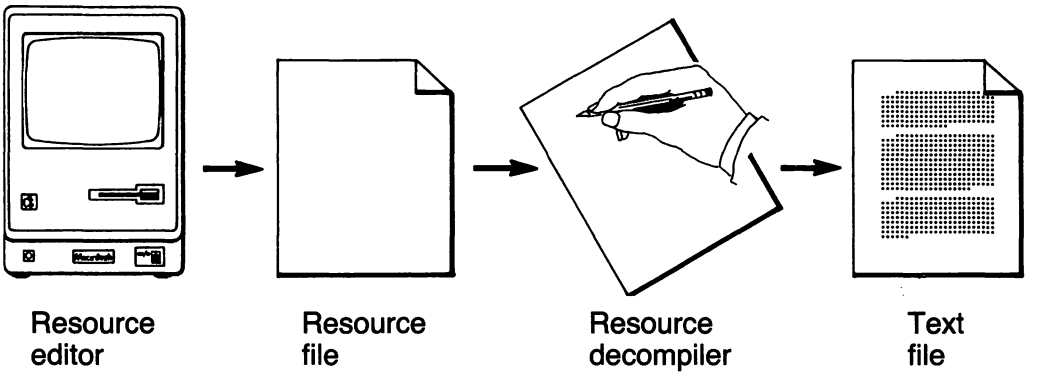


Figure 7-9. Resource decompiler steps.

compile. It's very straightforward and simpler than the programs in the following chapters, WHEREIS and BACKUP. Remember that the password is initially set to PassWord. This initial password is stored as a resource. And remember also to copy the program to all of your startup floppies, those with System and Finder, and install it with the SetStartup command. Then unless an uninvited visitor brings his or her own startup floppy, your system will be secure.

CHAPTER 8

Locating Lost Files

With 20 megabytes of space — enough for 10,000 pages — it's not hard to misplace or actually lose a file or folder. Perhaps you moved a document into a MiniFinder folder and neglected to return it. Perhaps to save time you filed a document in the current folder thinking you'd move it later and now can't remember which was the current folder that day. Perhaps you gave it what you thought was a memorable name, but now you aren't finding it quite so memorable. Through the ages people with hard disks have misplaced files, and that's why a file locator utility is a must for hard disk owners.

WHEREIS

This chapter provides a hard disk utility WHEREIS for locating files and folders. It's a short program written in Pascal, that you can use as is or add extra features to customize it.

When you double-click on the WHEREIS icon, you'll see a dialog box like the one in Figure 8-1. You specify the name of the file or folder you want to find, click the Find button, and WHEREIS will locate any matches for the file or folder you've specified in the dialog box (see Figure 8-2).

If it doesn't find a match, it displays the message shown in Figure 8-3. You can then either specify another file or folder and click the Find button or click the Quit button.

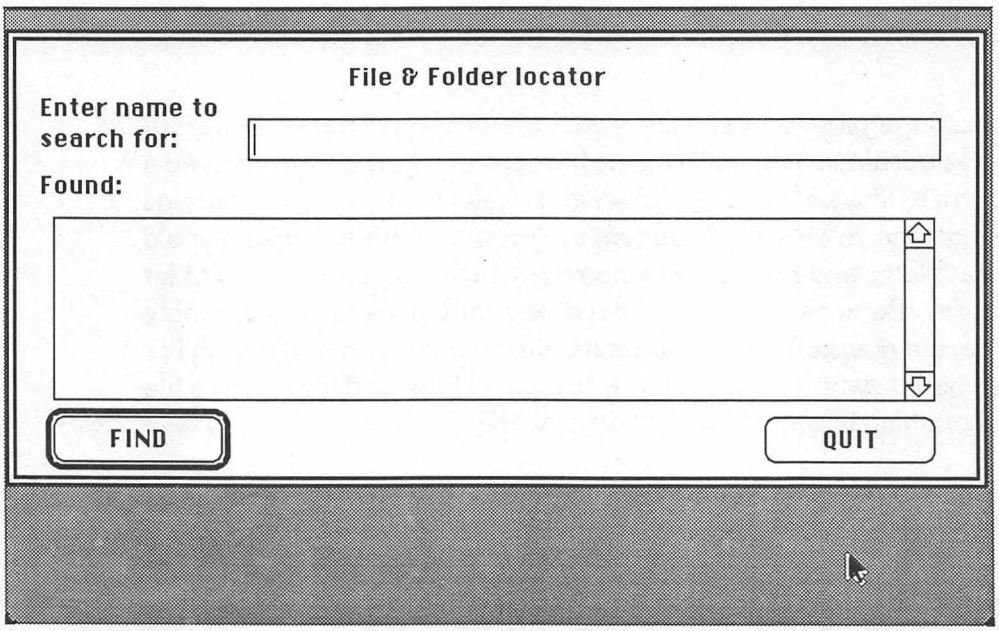


Figure 8-1. WHEREIS dialog box.

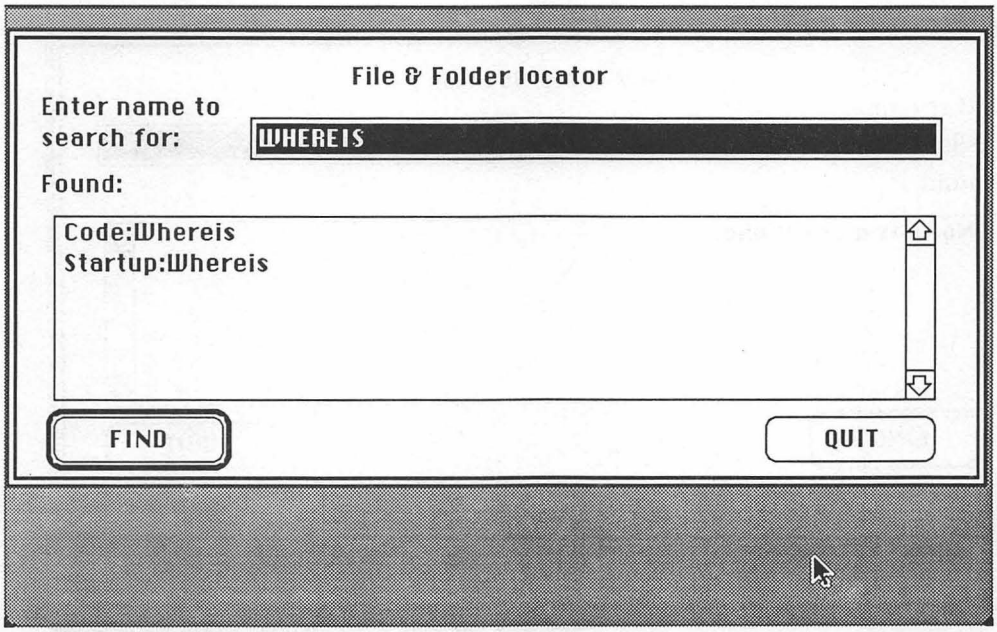


Figure 8-2. WHEREIS with list of files.

WHEREIS also lets you use a wildcard, the asterisk (*). If you can only remember the first few characters of a file or folder name, type the characters you remember followed by an asterisk. WHEREIS will find all files and folders that begin with the letters you've specified (see Figure 8-4).

If you simply press Return or click the Find button without specifying a file or folder name, WHEREIS will list every folder and file on your disk.

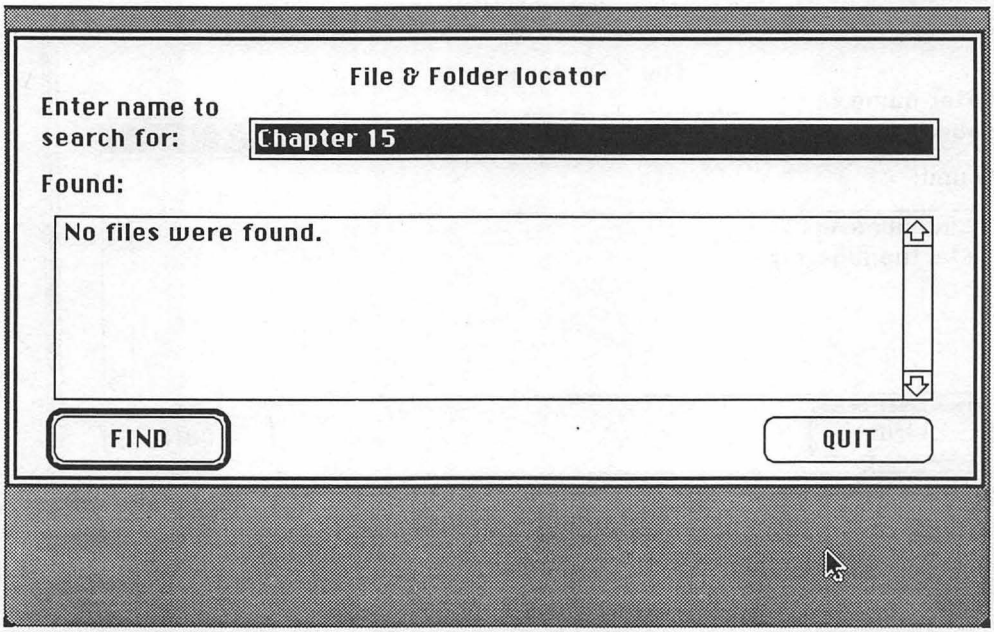


Figure 8-3. No files found message.

The Program

WHEREIS was written in MPW Pascal. We provide two lists of resources so that you can modify the program if you have MPW or TML Pascal. If you have another Pascal compiler such as TML, enter the program, but use the second list of resources. We've provided two lists. The MPW compiler has a unique way of doing resources. Most others use the second type of resource list.

If you've programmed before on a conventional computer such as the IBM PC, but this is your first Mac program, you may be surprised that the code looks somewhat different from what you're used to. Chapter

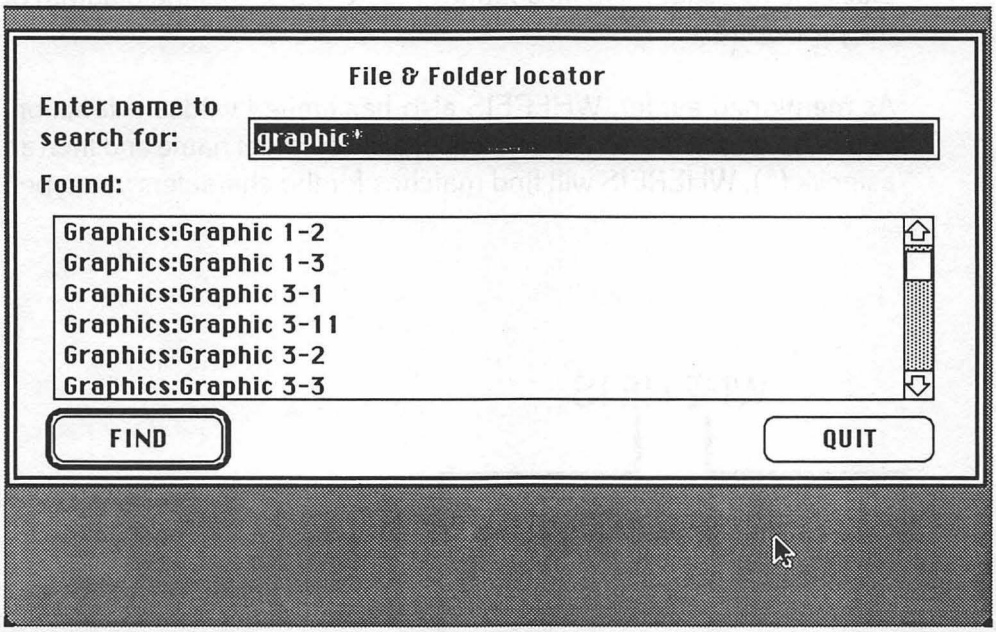


Figure 8-4. Wildcard example.

7 gives a short introduction to Mac programming and explains why the code looks different.

Overview

WHEREIS basically sets up dialog box, buttons, and callback routines. After the user types a search string (filename) and clicks Find, it starts with the root directory and compares the search string to the current filename. If it finds a match, it adds the filename to the list of files it displays in the list box. WHEREIS starts with the current directory, usually your hard drive, but also searches any floppy currently

inserted. If it completes its search and does not find a match, it displays the message, "No files found" Figure 8-5 shows the diagram of the procedure.

As mentioned earlier, WHEREIS also has limited wildcard function. You can type the first few characters of a file or folder name and then an asterisk (*). WHEREIS will find matches for the characters you type.

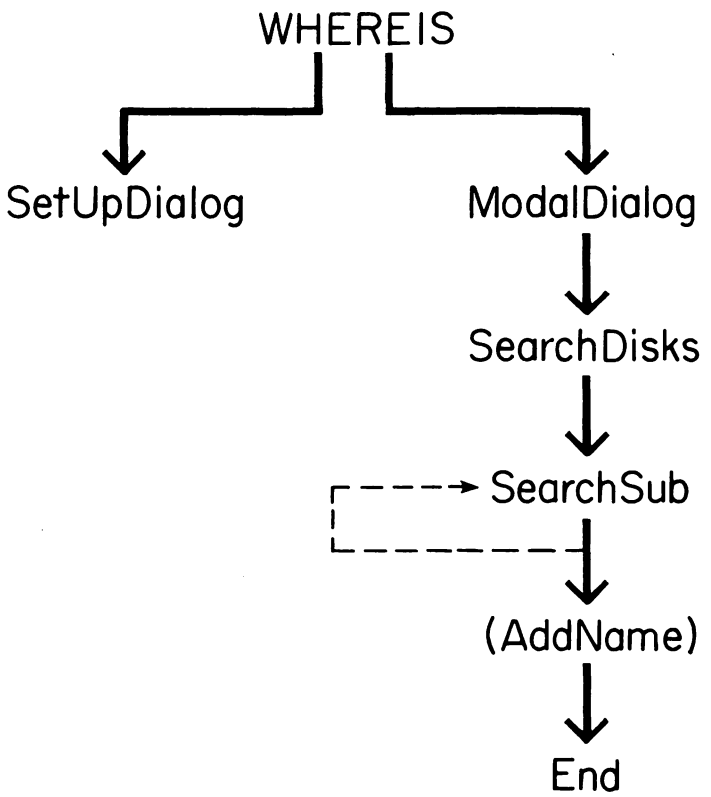


Figure 8-5. WHEREIS procedures.

Now let's look at the Pascal code for WHEREIS (see Figure 8-6a) procedure by procedure starting at the top to see how the program works.

Figure 8-6a.

[WHEREIS.LIST]

```
{   Whereis - A sample application written in MPW Pascal }
{       ©1987 - Bantam Books }
{       This program seaches the directories of all mounted }
{       diskettes for a given filename.  }

{ To make Whereis from MPW

    Pascal -s Whereis.P
    Rez -o Whereis.R.O Types.R Whereis.R
    Duplicate -y Whereis.R.O Whereis
    Link Whereis.P.O ∂
        MPW:PLibraries:PInterface.O ∂
        MPW:PLibraries:Interface.O ∂
        MPW:PLibraries:Paslib.O ∂
        MPW:Libraries:Runtime.O ∂
        -oc "????" -ot "APPL" -o Whereis
    SetFile Whereis -a B -c "HUH?"}


PROGRAM Whereis;

    USES
        {$LOAD pas.s}
        Memtypes,
        Quickdraw,
        OSIntf,
        ToolIntf,
        PackIntf;
```

```
CONST
    { Dialog item numbers }
    FindBtn = 1;        { The 'Find' button. }
    QuitBtn = 2;        { The 'Quit' button. }
    SearchName = 3; { The search filename editText. }
    FileList = 4;    { The userItem for the list of files found }
    ButtonFrame = 5; { The userItem for the frame
                                around the 'Find' button. }

    RtnKey = 13;        { The ascii value of the 'Return' key }

    NoFileStr = 100; { The resource ID for the
                                string "No files were found." }

    FSFCBLen = $03F6; { Used to determine which file
                                system is active (MFS or HFS). }

TYPE
    IntPtr  = ^INTEGER;


VAR
    theDialog : DialogPtr; { The pointer to the Dialog record. }
    theList : ListHandle; { The handle to the list of files found. }
    theScrlRect : Rect; { The rectangle enclosing the list's
                                scroll button. }
    itemHit : INTEGER; { Which button was pressed. }
    haveHFS : BOOLEAN;

{------------------------------  AddName  ------------------------------}

PROCEDURE AddName (theName: Str255);

    { Add a filename to the list }

    VAR
        newRow : INTEGER;
        tempLen : INTEGER;
        theCell : Cell;

    BEGIN
        newRow := LAddRow (1, 32767, theList);
        SetPt (theCell, 0, newRow);
```

```
        tempLen := LENGTH (theName);
        LSetCell (Ptr (LONGINT (@theName) + 1), tempLen, theCell, theList);
    END;

{------------------------------  SearchDisks  ------------------------------}

    PROCEDURE SearchDisks;

        { Search the directories for a given filename (searchStr).
          If searchStr is empty then show all files. }


        VAR
            myCPB : CInfoPBRec; { Used to read the information about
                                  a directory entry. }
            err         : OSErr; { Error code returned by file manager. }
            volName :  Str255; { Name of the current volume. }
            vRefNum : INTEGER;
            freeBytes : LONGINT;
            drQElPtr : QElemPtr; { Drive Queue element pointer. }
            fName  :  Str255; { File name of file being accesed. }
            tempStr  :  Str255; { Temp. string for wildcard search. }
            searchStr :  Str255; { Name of file being searched for. }
            searchLen : INTEGER;
            wildCard : BOOLEAN; { TRUE if this is a wildcard search. }
            itemType : INTEGER;
            itemHdl  : HANDLE;
            itemBox  : RECT;
            foundOne : BOOLEAN; { TRUE if any files were found. }

        PROCEDURE SearchSub (dirIDToSearch: LONGINT; curPathName : Str255);

            { The subroutine which does the actual searching. }


        VAR
            index : INTEGER;

        BEGIN { SearchSub }
            index := 1;
            REPEAT
                fName := ''; { Nil out name }
                myCPB.ioFDirIndex := index;
                myCPB.ioDrDirID := dirIDToSearch;
```

```
            { Get the info for the file }
            IF haveHFS THEN
                { Use PBGetCatInfo for HFS systems }
                err := PBGetCatInfo (@myCPB, FALSE)
            ELSE
                { Use PBHGetFInfo for MFS systems }
                err := PBHGetFInfo (@myCPB, FALSE);

            IF err = NoErr THEN
            BEGIN
                { Is it a wildcard search? }
                tempStr := fName;
                IF wildCard THEN
                    IF LENGTH (tempStr) > searchLen THEN
                        tempStr := COPY (fName, 1, searchLen);

                { Is there a match? }
                IF (searchStr = '') OR
                   EqualString (tempStr, searchStr, FALSE, FALSE) THEN
                BEGIN
                    AddName (CONCAT (curPathName, fName));
                    foundOne := TRUE;
                END;

                IF haveHFS THEN
                    { If the file is a directory then
                        search it before proceeding. }
                    IF BitTst (@myCPB.ioFlAttrib, 3) THEN
                    BEGIN
                        SearchSub (myCPB.ioDrDirID,
                                CONCAT (curPathName, fName, ':'));
                        err := 0;
                    END;
            END;
            index := index + 1;
        UNTIL err <> noErr;
    END;

BEGIN { SearchDisks }

    { Get the filename from the dialog }
    GetDItem (theDialog, SearchName, itemType, itemHdl, itemBox);
    GetIText (itemHdl, searchStr);
```

```
    wildCard := FALSE;
    searchLen := LENGTH (searchStr);
    IF searchLen > 0 THEN
    BEGIN
        IF searchStr[searchLen] = '*' THEN
        BEGIN
            wildCard := TRUE;
            DELETE (searchStr, searchLen, 1);
            searchLen := searchLen - 1;
        END;
    END;

    { Clear out the old names from the list }
    LDelRow (0, 0, theList);
    { We haven't found any files yet. }
    foundOne := FALSE;

    { Traverse the drive queue and search any mounted drives }
    drQElPtr := GetDrvQHdr^.QHead;

    WHILE drQElPtr <> NIL DO
    BEGIN
        err := GetVInfo (drQElPtr^.drvQElem.dQDrive,
                         @volName, vRefNum, freeBytes);
        IF err = noErr THEN
        BEGIN
            WITH myCPB DO
            BEGIN
                ioCompletion := NIL;
                ioNamePtr := @fName;
                ioVRefNum := vRefNum;
            END;
            SearchSub (2, CONCAT (volName, ':'));
        END;
        drQElPtr := drQElPtr^.drvQElem.qLink;
    END;
    { Were there no files found? }
    IF NOT foundOne THEN
        AddName (GetString (NoFileStr)^^);
END;
```

```
{------------------------------  ListUpdate  ------------------------------}

    PROCEDURE ListUpdate (theWindow: WindowPtr; theItem: INTEGER);

        { The update procedure for the file name list }

        VAR
            itemType : INTEGER;
            itemHdl  : HANDLE;
            itemBox  : RECT;

        BEGIN
            GetDItem (theWindow, FileList, itemType, itemHdl, itemBox);
            FrameRect (itemBox);
            LUpdate (theWindow^.visRgn, theList);
        END;

{------------------------------  ButtonUpdate  ------------------------------}

    PROCEDURE ButtonUpdate (theWindow: WindowPtr; theItem : INTEGER);

        { The update procedure for the ring around the 'Find' Button }

        VAR
            itemType : INTEGER;
            itemHdl  : HANDLE;
            itemBox  : RECT;
            oldPen   : PenState; { Used to save the current pen state }

        BEGIN
            GetPenState (oldPen);
            PenSize (3, 3);
            GetDItem (theDialog, ButtonFrame, itemType, itemHdl, itemBox);
            FrameRoundRect (itemBox, 18, 18);
            SetPenState (oldPen);
        END;


{------------------------------  SetUpDialog  ------------------------------}

    PROCEDURE SetUpDialog;

        { The setup procedure for the dialog. }
```

```
    VAR
        itemType : INTEGER;
        itemHdl  : HANDLE;
        itemBox  : RECT;
        dataBounds : RECT;
        rowNum   : INTEGER;
        newRow   : INTEGER;
        tempStr  :  Str255;
        tempLen  : INTEGER;
        theCell  : Cell;

    BEGIN
        { Create the dialog }
        theDialog := GetNewDialog (100, NIL, POINTER (-1));

        { Connect the update routine for
          the ring around the 'Find' Button. }
        GetDItem (theDialog, ButtonFrame, itemType, itemHdl, itemBox);
        SetDItem (theDialog, ButtonFrame, itemType, @ButtonUpdate, itemBox);

        { Connect the update routine for the File name list }
        GetDItem (theDialog, FileList, itemType, itemHdl, itemBox);
        SetDItem (theDialog, FileList, itemType, @ListUpdate, itemBox);

        { Set the rect for finding the scroll buttons. }
        InsetRect (itemBox, 1, 1);
        theScrlRect := itemBox;
        theScrlRect.left := theScrlRect.right - 15;

        { Create the File name list }
        itemBox.right := itemBox.right - 15;
        SetRect (dataBounds, 0, 0, 1, 0);
        theList := LNew (itemBox, dataBounds, Point (0), 0,
                         theDialog, TRUE, FALSE, FALSE, TRUE);
    END;

{------------------------------  Filter  ------------------------------}

    FUNCTION Filter (theDialog: DialogPtr; VAR theEvent: EventRecord; VAR itemHit: INTEGER):
BOOLEAN;

    { The filter procedure for the dialog }
```

```
    VAR
        temp : BOOLEAN;
        theChar :  Byte;

    BEGIN
        Filter := FALSE;

        { If there is no event pending then check for a
          disk inserted event to allow the system to mount disks. }
        IF theEvent.what = nullEvent THEN
            temp := GetNextEvent (diskMask, theEvent);
        CASE theEvent.what OF
            { Is it a mouseDown event? }
            mouseDown: BEGIN
                { Scroll the file list if the mouse is in theScrlRect }
                SetPort (theDialog);
                GlobalToLocal (theEvent.where);
                IF PtInRect (theEvent.where, theScrlRect) THEN
                BEGIN
                    temp := LClick (theEvent.where,
                                    theEvent.modifiers,  theList);
                    Filter := TRUE;
                    itemHit := 0;
                END;
                LocalToGlobal (theEvent.where);
            END;
            { Is it a keyDown event? }
            keyDown: BEGIN
                { Click the 'Find' button if they pressed the Return key }
                theChar := theEvent.message MOD 256;
                IF theChar = RtnKey THEN
                BEGIN
                    Filter := TRUE;
                    itemHit := FindBtn;
                END;
            END;
        END;
    END;

{------------------------------  Whereis  ------------------------------}

    BEGIN
        { Initialize the system }
        InitGraf (@thePort);
```

```
    InitFonts;
    FlushEvents (everyEvent, 0);
    InitWindows;
    InitMenus;
    TEInit;
    InitDialogs (NIL);
    InitCursor;

    { Set up the dialog }
    SetUpDialog;

    { Check to see which file system is provided }

    haveHFS := IntPtr (FSFCBLen)^ > 0;

    { Operate the dialog }
    REPEAT
        ModalDialog (@Filter, itemHit);

        { Did they hit the 'Find' Button? }
        IF itemHit = FindBtn THEN
        BEGIN
            { Display the 'Watch' Cursor }
            SetCursor (GetCursor (watchCursor)^^);

            { Search the mounted disks }
            SearchDisks;

            { Select the entire filename so that
                it can easily be re-entered. }
            SelIText (theDialog, SearchName, 0, 32767);

            { Re-display the 'Arrow' cursor }
            SetCursor (arrow);
        END;

    { Did they hit the 'Quit' Button? }
    UNTIL itemHit = QuitBtn;
    DisposDialog (theDialog);
END.
```

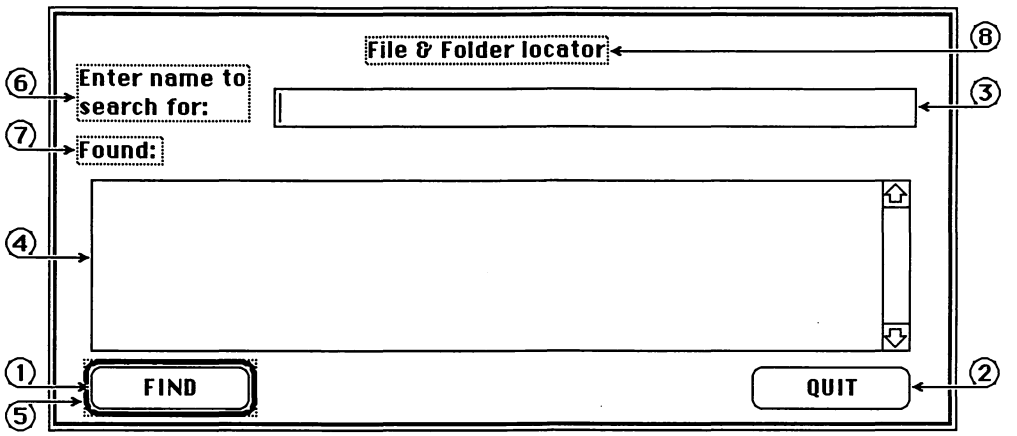
Declarations

The program starts with typical declarations, but they contain some unique information. The USES section lists the files to be included from the toolbox. Quickdraw, for example, is the group of functions that let you draw almost anything on the Mac screen. If you want to see which types and functions are included in these files, you can print them out. They contain Pascal source code. Next in the CONST section is a somewhat different group of constants. The first five items and the last item are resources for the resource compiler (see Figure 8-6b for the diagram of the resources). The numbers that follow these items in the CONST list are resource numbers for each item. For example, the Find button is resource 1, and the Quit button is resource 2. You can compare this list to the entries in the resource specification at the end of the program. The CONST RtnKey is simply the ASCII value for the Return key. It's used in the Filter procedure. The constant FSFCBLen is the address of the variable that tells us if the current disk we're searching is an HFS or MFS disk. It stores a -1 for MFS disks and a positive number for HFS disks. The VAR section contains your standard global variables.

AddName

This procedure is called by the following procedure, SearchDisks, when it finds a match for a specified filename or when it's searched the entire disk and found no matches. AddName takes a filename or the string "No files found" and sets it up so it can be added to the list in the list box. The filename includes the pathname, that is the names of the folders the file is nested inside of.

Three lines of this procedure contain calls to ROM functions. They are LAddRow, SetPt, and LSetCell. This procedure does not actually write the new filename in the list box on the screen. Instead it writes it into an update region, and then notifies the Event Manager that the update area has changed. Then when ModalDialog, the ROM dialog box manager, checks with the Event Manager and sees it has work to do, it calls



Item 1: The 'Find' Button.
Item 2: The 'Quit' Button.
Item 3: Search file name EditText.
Item 4: The List of files found (UserItem).
Item 5: The ring around the 'Find' button (UserItem).
Item 6: 'Enter name to search for' StaticText.
Item 7: 'Found:' StaticText.
Item 8: 'File & Folder locator' StaticText.

The Whereis Icon

Figure 8-6b.

ListUpdate, a procedure you'll see later in the program, and actually takes care of putting the filename on the screen. Although this may seem rather convoluted, it really is a handy way of doing several things and not having to worry about redrawing the screen after each one.

If you decide you would like to display the files WHEREIS finds differently, for example to put each folder name on a separate line and

indent nested folders beneath parent folders, this is the place to add the code to do that.

SearchDisks

This is the code that does the major work for this program. It takes the search string, checks for wildcards, and allocates a portion of memory for the filename and volume number. Then it calls on the nested recursive procedure SearchSub which searches through every subdirectory on the disk.

SearchSub sets a pointer to the first directory and begins with a repeat loop with this line:

```
err := PBGetCatInfo (@myCPB, FALSE)
```

The ROM call PBGetCatInfo gets the first filename in the current directory and returns an error value of zero, no error, as long as there is another file in the list. Next SearchSub gets the correct information for the folder or file depending on whether we have an HFS or MFS disk. Then SearchSub looks to see if there was a wildcard. If yes, it copies only the portion of the filename with the same number of characters as those in the search string into a temporary variable, tempStr. Then it calls the Pascal function EqualString to compare tempStr and searchStr. If it finds a match, it calls the previous procedure AddName. Then if the filename just checked was a folder, SearchSub recursively calls itself until it reaches the end of the files in the folder. It continues searching through folders and files until it reaches the end of the directory listing. When it reaches the end of the list, it returns an Err value other than zero, and returns to the main procedure, SearchDisks.

At this point SearchDisks checks to see if there are any more drives to search. Then after the entire search is complete, if no matches were found, it calls AddName with the "No files found" string.

ListUpdate and ButtonUpdate

These are two procedures that the following procedure SetUpDialog hooks up. ListUpdate is called by the List Manager and is used to draw the contents of the list on the screen. Actually the ModalDialog call in the main program calls the List Manager whenever there is an item to add to the list. ButtonUpdate uses ROM calls to draw the extra ring around the Find button. This ring indicates that the Find button is the default.

SetUpDialog

This procedure creates the dialog box, connects to the ButtonUpdate and ListUpdate procedures so when the system gets an update event, it will update the list and the ring around the Find button. It also sets up the scroll bar, and sets up the necessary parts for the list of filenames.

Filter

Filter checks for events that ModalDialog in the main program doesn't cover. These include a carriage return inside a text box — ModalDialog would interpret this as a carriage return instead of the equivalent of pressing the Find button — and a click inside of a user item such as the scroll bars. When either of these events happen, Filter takes the appropriate action.

WHEREIS

WHEREIS is the main program. It first calls the initialization routines, then sets up the dialog box, and checks to see if we have an HFS or MFS disk. Then it calls ModalDialog. ModalDialog is the program's dialog box controller. It activates the dialog box, checks for events

such as keypresses, mouse clicks, and updates and then takes appropriate action.

ModalDialog uses the Filter procedure when needed and returns a value in itemHit if the Find or Quit buttons were pressed. If the Find button was pressed, it changes the cursor to the watch and calls SearchDisks. When SearchDisks finishes it uses the ROM SelIText to select the filename. This way after you finish one search if you want to enter another file or folder name, you won't first have to select the text. Finally it replaces the watch cursor with the arrow. This sequence is repeated until the user hits the Quit button. Just before the program ends, we clean up with the toolbox procedure, DisposeDialog.

RESOURCE LISTS

There is an additional part to the program listing. It's the resource specification for WHEREIS. Figure 8-7 shows the resource list in MPW Pascal and Figure 8-8 shows an equivalent resource list for TML Pascal.

Figure 8-7.

[MPWResources.Whereis]

```
/* The resource specification for the application 'Whereis'. */
/* Presented in MPW resource compiler format */

resource 'DLOG' (100, preload) {
        {40, 10, 260, 500},          /* Window's BoundsRect */
        dBoxProc,                    /* Window's ProcID */
        visible,
        noGoAway,
        0x0,                         /* Window's RefCon (unused) */
```

```
        100,                            /* Resource ID of DITL */
        ""                              /* Dialog title   (unused)*/
};

resource 'DITL' (100, preload) {
        {       /* array DITLarray: 8 elements */
                /* [1] 'FIND' button. */
                {192, 16, 216, 104},
                Button {
                        enabled,
                        "FIND"
                };
                /* [2] 'QUIT' button. */
                {192, 384, 216, 472},
                Button {
                        enabled,
                        "QUIT"
                };
                /* [3] Filename to search for. */
                {40, 120, 56, 472},
                EditText {
                        enabled,
                        ""
                };
                /* [4] List of files found. */
                {88, 16, 184, 472},
                UserItem {
                        disabled
                };
                /* [5] Ring around 'FIND' button. */
                {188, 12, 220, 108},
                UserItem {
                        disabled
                };
                /*  [6]  */
                {24, 8, 56, 104},
                StaticText {
                        disabled,
                        "Enter name to search for:"
                };
                /*  [7]  */
                {64, 8, 80, 56},
```

```
                StaticText {
                        disabled,
                        "Found:"
                };
                /*  [8]  */
                {8, 168, 24, 304},
                StaticText {
                        disabled,
                        "File & Folder locator"
                }
        }
};

/* The 'No files were found.' string. */
resource 'STR ' (100, preload) {
        "No files were found."
};

/*      The following resources are used to attach an
                icon to Whereis. In order to use this icon,
                Whereis' creator has to be set to 'HUH?' and
                its bundle bit has to be set.  The standard
                application icon will be used if these
                resources are omitted. */

/* The finder bundle. */
resource 'BNDL' (128, purgeable) {
        'HUH?',
        0,
        {      /* array TypeArray: 2 elements */
                /*  [1]  */
                'ICN#',
                {      /* array IDArray: 1 elements */
                        /*  [1]  */
                        0,
                        128
                };
                /*  [2]  */
                'FREF',
                {      /* array IDArray: 1 elements */
                        /*  [1]  */
                        0,
```

```
                    128
                }
        }
};

/* The finder icon. */
resource 'ICN#' (128, purgeable) {
        {       /* array: 2 elements */
                /* [1] */
                $"011F 0000 02A0 FF00 0460 0080 0820 0080"
                $"1010 0080 2008 0080 4074 00F8 808A 008C"
                $"41F4 00AA 206F FF8F 101E 0271 0826 0201"
                $"FE43 C2FD 8387 E201 BA86 62F1 83C0 6201"
                $"BC40 C2FD 8041 8201 BF41 83FF 8040 0220"
                $"BE41 8410 8041 8808 B7C0 1004 883F E01A"
                $"F800 4025 0800 207A 0800 3017 0800 280F"
                $"0800 2413 0800 2220 0800 2140 0FFF E080";
                /* [2] */
                $"011F 0000 03BF FF00 07FF FF80 0FFF FF80"
                $"1FFF FF80 3FFF FF80 7FFF FFF8 FFFF FFFC"
                $"7FFF FFFE 3FFF FFFF 1FFE 03FF 0FE6 03FF"
                $"FFC0 03FF FF80 03FF FF80 03FF FFC0 03FF"
                $"FFC0 03FF FFC0 03FF FFC0 03FF FFC0 03E0"
                $"FFC0 07F0 FFC0 0FF8 FFC0 1FFC FFFF FFFE"
                $"FFFF FFFF 0FFF FFFE 0FFF FFFF 0FFF EFFF"
                $"0FFF E7F3 0FFF E3E0 0FFF E1C0 0FFF E080"
        }
};

/* The finder file reference. */
resource 'FREF' (128, purgeable) {
        'APPL',
        0,
        ""
};

/* My 'Brag' string (required by the finder). */
resource 'HUH?' (0, purgeable) {
        "Whereis - Written by Nancy Andrews and Ron Aldrich - August, 1986"
};
```

Figure 8-8.

[TMLResources.Whereis]

```
* The resource specification for the application 'Whereis'.
* Presented in MDS RMaker format (used by TML Pascal).

Whereis.Rel

* The dialog.
TYPE DLOG
        ,100  (4)                       ;; Resource ID
Unused Title                    ;; Dialog title (unused)
40 10 260 500                   ;; Window's BoundsRect
Visible NoGoAway
1                               ;; Window's ProcID
0                               ;; Window's RefCon (unused)
100                             ;; Resource ID of DITL

* The dialog's item list.
Type DITL
        ,100  (4)                       ;; Resource ID
8                               ;; # of items in list

Button Enabled                  ;; Item # 1 - 'FIND' button
192 16 216 104
FIND

Button Enabled                  ;; Item # 2 - 'QUIT' button
192 384 216 472
QUIT

EditText Enabled                        ;; Item # 3 - Filename to search for
40 120 56 472

UserItem Disabled               ;; Item # 4 - List of files found
88 16 184 472

UserItem Disabled               ;; Item # 5 - Ring around 'FIND' button
188 12 220  108
```

```
StaticText Disabled                ;; Item # 6
24 8 56 104
Enter name to search for:

StaticText Disabled                ;; Item # 7
64 8 80 56
Found:

StaticText Disabled                ;; Item # 8
8 168 24 304
File & Folder locator


* The 'No files were found.' string.
TYPE STR                           ;; 'STR ' (Space is required.)
        ,100  (4)
No files were found.

* The following resources are used to attach an
*    icon to Whereis. In order to use this icon,
*    Whereis' creator has to be set to 'HUH?' and
*    its bundle bit has to be set.  The standard
*    application icon will be used if these
*     resources are omitted.

* The finder bundle.
Type BNDL
        ,128  (32)
HUH? 0
ICN#
0 128
FREF
0 128

* The finder icon
Type ICN# = GNRL
        ,128  (32)
.H
011F 0000 02A0 FF00 0460 0080 0820 0080          ;; Icon data
1010 0080 2008 0080 4074 00F8 808A 008C
41F4 00AA 206F FF8F 101E 0271 0826 0201
FE43 C2FD 8387 E201 BA86 62F1 83C0 6201
BC40 C2FD 8041 8201 BF41 83FF 8040 0220
```

```
BE41 8410 8041 8808 B7C0 1004 883F E01A
F800 4025 0800 207A 0800 3017 0800 280F
0800 2413 0800 2220 0800 2140 0FFF E080
011F 0000 03BF FF00 07FF FF80 0FFF FF80          ;; Mask data
1FFF FF80 3FFF FF80 7FFF FFF8 FFFF FFFC
7FFF FFFE 3FFF FFFF 1FFE 03FF 0FE6 03FF
FFC0 03FF FF80 03FF FF80 03FF FFC0 03FF
FFC0 03FF FFC0 03FF FFC0 03FF FFC0 03E0
FFC0 07F0 FFC0 0FF8 FFC0 1FFC FFFF FFFE
FFFF FFFF 0FFF FFFE 0FFF FFFF 0FFF EFFF
0FFF E7F3 0FFF E3E0 0FFF E1C0 0FFF E080

* The finder file reference
Type FREF
        ,128  (32)
APPL 0

* My 'Brag' string (required by the finder).
Type HUH? = STR
        ,0  (32)
Whereis - Written by Nancy Andrews and Ron Aldrich - August, 1986
```

This list of resources is comparable to records with preset data — records to draw things such as the buttons and items in the dialog box. The part of the ROM called the Resource Manager provides access to the list of resources. Resources 1 to 5 correspond to the resources named in the CONST declarations of the program. Resources 6, 7, and 8 are text strings, and the last resource is the "No files found" string.

Next is the code for the WHEREIS icon, an icon made with the resource editor's icon editor. You use this editor to create an icon much like you use "Fat Bits" in Paint. Then you decompile it and get the code listed at the end of the resource list.

As you can see, all hard work is done by ROM routines — drawing the

parts of the screen, updating the list box, monitoring keypresses and mouse moves. The two complexities of this program are the "callbacks", the fact that the operating system has control over routines in your program and can call them when needed, and recursion. The SearchSub procedure recursively calls itself until it comes to the end of the directory listing.

ENHANCEMENTS

This program is a good start for a file locator. But there are two additional features you may wish to add. One thing you might consider is adding an initial wildcard feature. Currently the program allows a wildcard (*) at the end of a string, allowing you to search for all files that begin with the same character or characters. You might want to also be able to search for all files with the same ending. For example, if you have used the PackIt III program to compress files and named all compressed files with the .pit extension, you may wish to search for all your compressed files by specifying *.pit. To add this capability this is what to do.

First you'll need to change the VAR section and make two wildcard variables, wildCard_1 and wildCard_2. They are both Boolean variables. Then you'll need to modify the Wildcard section of SearchDisks to check for an asterisk at the beginning as well as at the end. Change the line that checks for an asterisk at the end to wildCard_1: = true. Then after the check for the asterisk at the end, add the these lines to check for the asterisk at the beginning.

```
If searchStr[1] = '*' THEN
    wildCard_2 := TRUE;
    DELETE (searchStr, 1,1);
        searchLen := searchLen - 1;
```

You'll also have to change the existing wildCard in SearchSub to wildCard_1 and add these lines for a wildCard_2.

```
IF wildCard_2 AND LENGTH (tempStr) > searchLen then
    BEGIN
      tempLen := LENGTH(tempStr) - searchLen + 1;
      tempStr := COPY(fName, tempLen, LENGTH(tempStr));
    END;
```

This copies the last few characters of the filename, as many as are in the search string, into the variable tempStr. Then the EqualString function compares the two strings as before. You may notice one problem with this. I used a local variable that's not declared, so add the integer, tempLen, to the VAR list for this procedure. With these few lines you have the additional feature of a wildcard at the beginning as well as at the end.

Another feature you might wish to add is to have WHEREIS display a list of files of a certain type, for example all your Excel worksheets or all text files. This is a more complex problem. Basically, this is how you'd approach it. The filename contains a four-character ID tag that tells you the type of a particular file. You need to find out which files use which identifiers. I'll give you the code you can add to the WHEREIS program so it will print the four-character file type tag in square brackets after the file name. Once you can get this information, it will be an easy Pascal exercise to search for files with the ID type you specify.

First, you'll need to add these lines to the IF statement that prints the pathName and filename for WHEREIS matches. The first line calls a new procedure called OSTypetoString. It has two parameters. The first is a call to myCPB, the File Manager record that stores information about each file, the same one we used earlier to get the filename. This time we're extracting file type information. The catch is that this file

type is called OSType, and in order to display it and compare it to a string we have to coerce it into a string type. So the procedure OSTypeToString takes this information and converts it to a string which it returns in the second parameter, tempStr. The second line changes the call to AddName. In addition to printing the pathName and filename, it tells AddName to print the file type surrounded by square brackets.

```
IF (searchStr = '') OR EqualString . . .
BEGIN
    OSTypeToString (myCPB.ioF1FndrInfo.fdType, tempStr);
    AddName(CONCAT(curPathName,fName,'[',tempStr,']');
    . . .
END

Procedure OSTypeToString (OS : OSType; VAR IDString : Str255);

VAR
    fileType : RECORD
        CASE INTEGER OF
            0:    (str : Str255)
            1:    (filler : INTEGER;
                   OST : OSType);
        END;
BEGIN
  fileType.OST := OS;
  fileType.str[0] := chr(5);
  Delete(fileType.Str,1,1)
  IDString := fileType.Str;
END;
```

Next is the procedure to change the file type from OSType to a Pascal string. To do this we use a variant record called fileType. The first line of the procedure puts the file type into the record variant field called fileType.OST. This is an OSType variable. Then we look at this four-character area through the string variant part of the record and make a few changes so it has the requisite parts of a Pascal string. The next line adds the string length. Pascal strings store the length in the first byte of the string. Then the 68000 has a quirk that says that OSTypes must start at an even address. This means that after the length byte in the string, there is one byte of garbage and then the four characters we want. To get rid of the garbage the next line uses the Pascal Delete function to delete the second byte and move the four characters over immediately after the length byte. Finally, we have the string in the form we want and all we need to do is assign it to the variable IDString and pass it back to the main program.

The next step is to preface the filename the user types in with something like a wildcard, perhaps a caret (^), so you'd know this search was for file types rather than filenames. Then you'd have to add the code to SearchSub to look for a match for the file ID type rather than for the file or folder name if the first character was a caret (^). We'll leave these two parts as an "exercise left for the reader . . . ". They're straightforward; you can do them with standard Pascal without any additional toolbox calls.

If you haven't written many Mac program before, all this may look pretty complicated. And it's only fair to mention that you could write this program more easily or at least more conventionally without using the toolbox very much and without incorporating the standard Mac visual interface. You could basically write a text program that began by writing a prompt on the screen — type name of file you want to find. Then the program could do the search, display list of matches without a dialog box, and repeat the prompt. But that isn't really Mac programming and with all the tools available, to circumvent them seems like not playing fair. So instead what you have here is an example of a more complicated, but definitely Mac, program.

CHAPTER 9

Backing Up

WHY BACKING UP

People will do almost anything to avoid backing up because it is so utterly boring. Inserting floppy after floppy and then fidgeting while 20 or more megabytes of hard disk data are copied to floppies seems like an incredible waste of time. But not backing up can be costly. Imagine these scenarios. You've most likely heard them before or have an equally good one of your own to add.

. . . You're a PhD candidate in biology doing hunger research for your doctoral dissertation. You have a year's worth of data from injecting and weighing rats stored on the hard disk and you didn't bother to back it up. The computer crashed, the rat data was gone, and you were back to the beginning of the rat race.

. . . It was a dark and stormy night. You had been working for hours on

your novel, which would transform the face of fiction for all time. Each sentence was more lyrical than the last. But suddenly, the roar of a lightening bolt announces that you're going to lose power. Some hours later power is restored. You turn on your Mac only to await the collapse of the literary world — the file with your novel is gone.

Backing up is boring, but so is putting money in the bank and buying life insurance. And what backing up does is ensure that you won't lose valuable data from system failures, power outages, or human errors.

Some hard disk manufacturers such as General Computer and Peripheral Land include a backup utility with the hard disks they sell. These are covered in chapter 2. Others such as Apple do not. If you're fortunate enough to have a hard disk with a backup utility, use it. The ones I tested worked well.

THE BACKUP PROGRAM

The rest of this chapter is for those of you who have neither a floppy or tape backup system and are feeling guilty or scared enough to want to back up regularly without the hassle of doing it by hand, selecting and dragging folder after folder to floppy after floppy. It presents what we call "An Amazing Symmetric Backup Program". You can use it both to backup files from a hard disk to floppies or to another hard disk or to restore files from floppies or a second hard disk to the primary hard disk.

BACKUP is a symmetric backup program because it uses the same code when it backups and when it restores. It works the same whether you're copying from a hard disk to floppies or from floppies to a hard disk. And it's amazing because it doesn't change the file and folder structure on the destination disk. If your hard disk is broken, you can use the files you've backed up right from the floppy disk. If you need to restore just one file you've accidentally erased, you can insert the floppy and use it without the BACKUP program just as you would any

floppy — locate the file you need and drag it to the hard disk. If you've had to reformat your hard disk and want to completely restore everything, you simply run BACKUP backwards — that is you copy from the floppies back to the hard disk. Both the files and folders will be recreated and nested exactly as they were when you backed them up.

We wanted to be able to print the source code for the program in this book. A program listing of more than 1,000 lines is difficult to read and use in book format. The backup program listed here stretches that limit; it's about 1,040 lines long. But it's short for a backup program. The compiled version of our backup program is 10K; the compiled version of HyperDrive's backup program is 144K.

The BACKUP program does have two minor limitations. It will only backup to 800K floppies — 400K floppies will not work. And if a file is larger than 800K, it won't be backed up. You'll see a message that explains that the file is too large to backup and the program will continue.

What It Does

This is what the BACKUP program does. First you click the Source disk button to choose the disk to backup. Then click the Destination button to indicate where you want your files copied. This will usually be a floppy drive unless you are fortunate to have another hard disk to use for backup purposes. Next you choose the extent of files to backup from a menu of dates. Then you can choose either Dry Run or the actual backup from the File menu. Dry Run looks at the number of files to back up based on the date set in the Backup last menu, tells you how many files and folders there are and how many floppy disks you'll need but does no actual copying of files (see Figure 9-1).

When you choose BACKUP from the menu, it prompts you to insert a

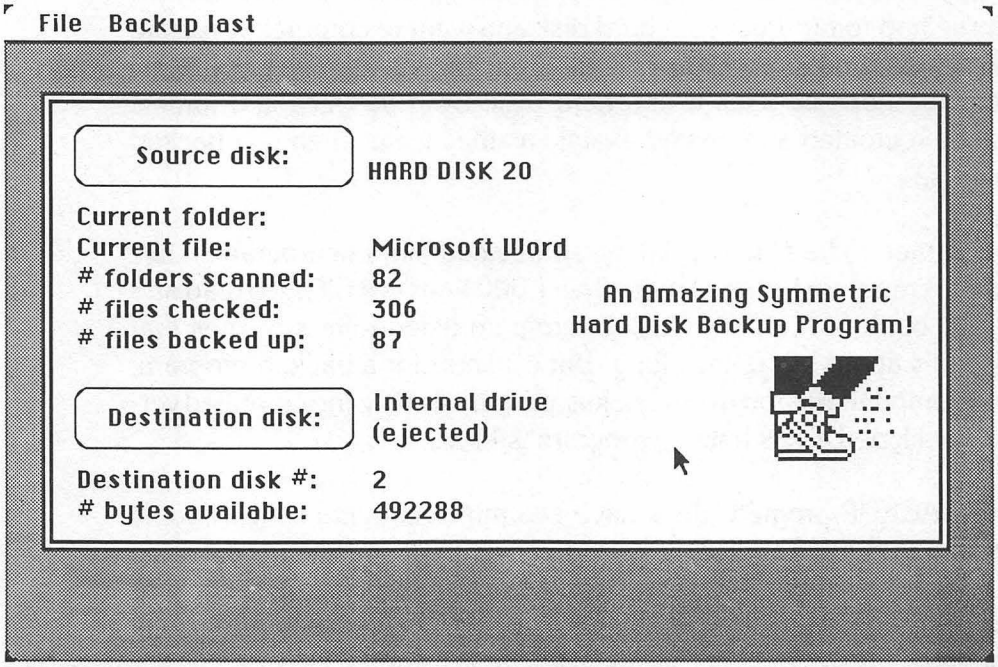


Figure 9-1. Backup after Dry Run.

floppy if your destination is a floppy drive. If it's not initialized, it asks if you want to initialize. If you do, it formats it for you, and then copies files and folders. If the disk you insert is not blank, it uses the available space. The only files it will replace are ones with exactly the same names. When the current floppy is full, it prompts you for the next one.

At any time during the backup, you can press Command-period to cancel the backup (see Figure 9-2). You can also choose Eject from the File menu. You'd use this to eject a floppy currently inserted if it was not one you wanted to use to start your backup.

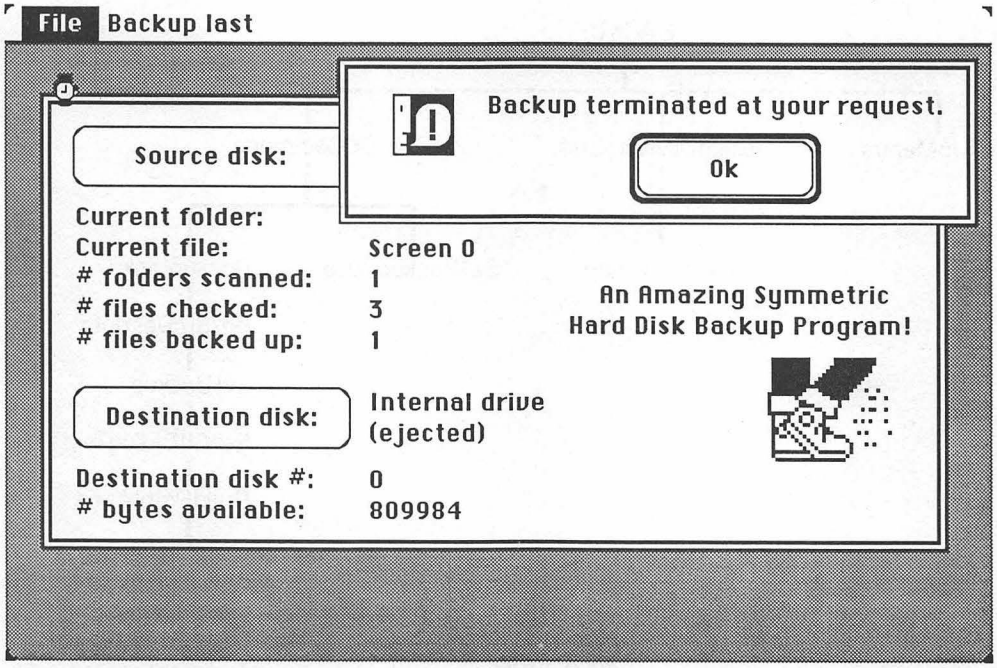


Figure 9-2. Abort alert box.

How It Works

If you've looked at the LOCKIT and WHEREIS programs in chapters 7 and 8, you'll notice that this program adds menus. You can look at the menu code and resources to see how menus can be programmed, and if you wish, add options or customize the menus to better suit your needs.

Figure 9-3 gives an overview of how the backup program works. First, BACKUP sets up the dialog box and draws the menus. Then just as in WHEREIS and LOCKIT ModalDialog takes over, uses the Filter procedure and the variable itemHit to give the Event Manager the information it needs to call routines in our program. It takes appropriate action when a menu selection is made or a drive button is pressed. If a

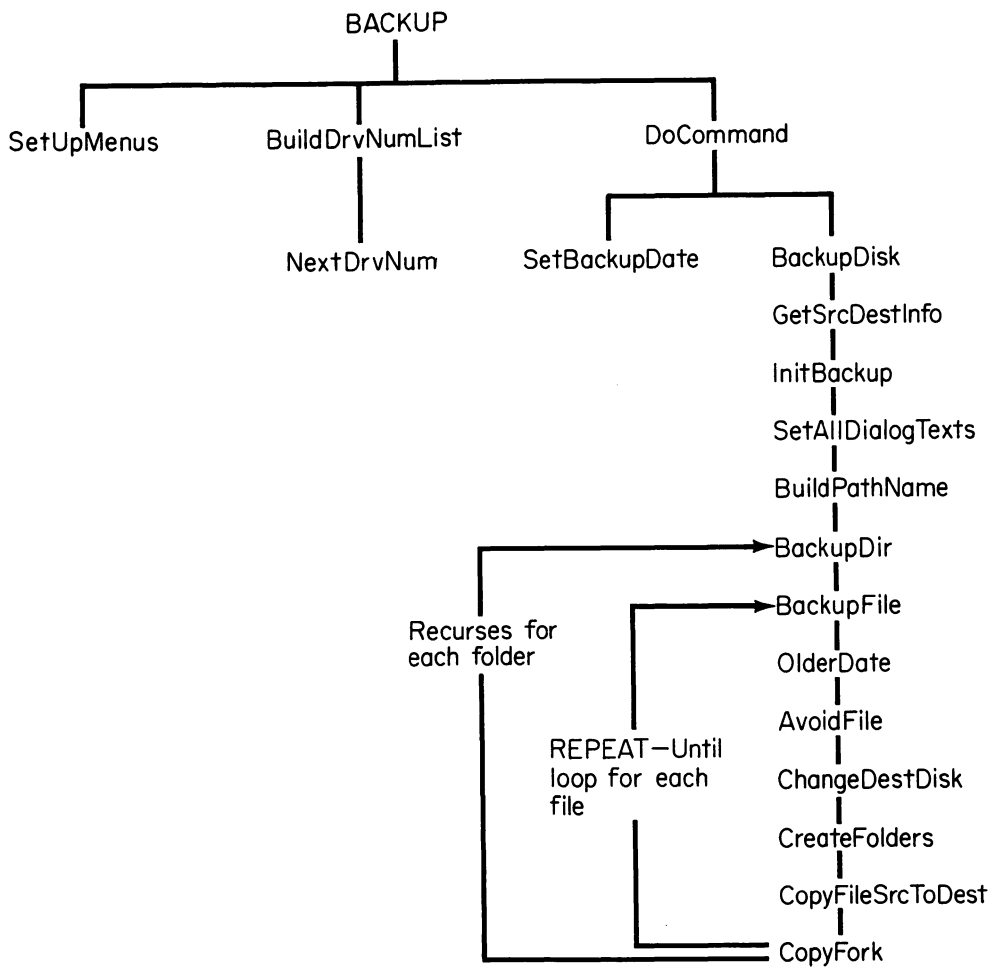


Figure 9-3. Backup diagram.

drive button is pressed, it updates the source or destination information on the screen. If a selection is made from the Backup last menu, ModalDialog calls DoCommand which uses SetBackupDate. If a selection from the File menu is made, ModalDialog again calls

DoCommand. If the command is Backup, DoCommand calls BackupDisk. BackupDisk calls BackupDir. BackupDir lines up the files in the current directory, then calls BackupFile which goes through a Repeat-Until loop for each file. It first calls OlderDate to compare the file's date to the date chosen on the Backup last menu. If it is one that meets the date criterion and consequently needs to be backed up, it calls CreateFolders and then CopyFileSrcToDest which calls the nested procedure CopyFork twice, once to copy the file's data fork and again to copy its resource fork. When all the files in a folder are copied, it returns to BackupDir which recursively calls itself for each folder on the disk. If Dry Run rather than Backup is chosen from the menu, it goes through the procedures in much the same way but skips the actual copying with CopyFileSrcToDest and CopyFork.

BACKUP LISTING

Now lets look at the program starting with the declarations and then procedure by procedure (see Figure 9-4).

Declarations

The USES section, as in the previous programs, lists the toolbox routines the program uses. Constants include the resources for the dialog and alert boxes, drive buttons, and menus. There are also key codes for the Command key and period and for the Return key. The VAR sections consist mainly of variables for information about the source disk — things like drive number, volume name, folder name and filename — and similar information about the destination disk. It also includes a section of "dummy" variables that are used when a function returns a value we don't need.

Figure 9-4.

[BACKUP.LIST]

```
{
    Backup - A small hard disk backup program
    ©1987 - Bantam Books

    To make the Backup program from MPW:

    Pascal -P "backup".p
    Rez -o "backup".r.o Types.R "backup".r

    Duplicate -y backup.r.o backup
    Link backup.p.o ∂
        "HARD DISK 20:MPW:PLibraries:"PInterface.o ∂
        "HARD DISK 20:MPW:PLibraries:"Paslib.o ∂
        "HARD DISK 20:MPW:Libraries:"Runtime.o ∂
        -oc "????" -ot "APPL" -o backup
}

PROGRAM Backup;

    USES
        {$LOAD backup.s}
        Memtypes, Quickdraw, OSIntf, ToolIntf, PackIntf;

    CONST
        BUFLEN     = 8192;    {#Bytes for file copy buffer}

        MaxFileSize = 800000; {Longest file must fit on an 800K disk}

        {resource IDs/menu IDs for menus}
        fileID     = 129;
        backupID = 130;

        {index for each menu in myMenus (array of menu handles)}
        fileM      = 1;
        backupM  = 2;

        menuCount = 2;    {total number of menus}
```

```
{Commands in the File menu:}
BackupCmd = 1;
DryRunCmd = 2;
EjectCmd = 3;
QuitCmd  = 5;

{Backup time commands from "Backup last" menu:}
Hour1Cmd = 1;
Hour8Cmd = 2;
Hour24Cmd = 3;
Hour48Cmd = 4;

Week1Cmd = 5;
Week2Cmd = 6;
Month1Cmd = 7;
Month2Cmd = 8;
Year1Cmd = 9;
WayBackCmd = 10;

{Source and destination selector buttons from the main dialog box:}
SrcDrvBtn = 2;
DestDrvBtn = 14;

{Names of string display items main dialog:}
SrcVolText = 3;
PathText = 5;
SrcNameText = 7;
NumFoldersText = 9;
NumFilesText = 11;
NumFilesCopText = 13;
DestVolText = 15;
FreeBytesText = 17;
NumDestDisksText= 19;
SrcDrvText = 21;
DestDrvText = 22;

{Resources for Change Disk alert box:}
AbortBtn = 1;
DiskChangedOk = 999;

{Key codes:}
```

```
    RtnKey      = 13;
    PeriodKey = 46;
    CmdKeyBit = 8;

TYPE
    Str32       =  String[32];

VAR
    myMenus      : ARRAY [1..menuCount] OF MenuHandle;

    {Date/time of oldest file to back up}
    backupDate    : LONGINT;

    {Global configuration flags:}
    doneFlag    : BOOLEAN;
    dryRunFlag    : BOOLEAN;

    theDialog     : DialogPtr; { The pointer main Dialog record.}
    itemHit      : INTEGER; { Which main dialog button was pressed.
                                        (actually we don't have any buttons
                                        in the main dialog!) }

    {Buffer for file copying:}
    copyBuf      : ARRAY [0..BUFLEN] OF CHAR;


    {Information about all disk drives connected to the system:}
    numDrives    : INTEGER;
    drvNumList     : ARRAY [1..20] OF INTEGER;

    {Variables for the backup "source" - the hard disk being backed up:}
    srcDrvNum    : INTEGER; {drive number}
    srcVolName      : Str255; {volume name}
    srcVRefNum    : INTEGER; {volume reference number}
    srcFreeBytes    : LONGINT; {free space}
    srcCPB      : CInfoPBRec; {param block used for low-level I/O}
    srcFName      : Str255; {current file being checked}
    srcFLen      : LONGINT; {#bytes in file being checked}

    {Variables for the backup "destination" - the floppy being copied onto.
     NOTE: The user automatically selects which floppy to use by simply
     inserting a disk in one drive or the other when prompted for a new
     disk.}
```

```
    newDestReady   : BOOLEAN; {TRUE if there's a good disk in dest drive}
    destDrvNum     : INTEGER; {number of floppy drive}
    destVolName    : Str255; {volume name}
    destVRefNum    : INTEGER; {volume ref number}
    destFreeBytes  : LONGINT; {free space left on floppy}

    {Variables for the the current folder path, used on both src and dest:}
    pathList    : ARRAY [1..30] OF Str32;
    pathLen     : INTEGER;
    pathName    : Str255;

    {Backup statistics:}
    numFoldersScanned: INTEGER;
    numFilesScanned: INTEGER;
    numFilesCopied : INTEGER;
    numDestDisks   : INTEGER;

    {Dummy variables to store unwanted function results:}
    iDummy    : INTEGER;
    oDummy    : OsErr;
    bDummy    : BOOLEAN;

PROCEDURE Debug; INLINE $A9FF;

{---------------------------  OlderDate  ----------------------------}

FUNCTION OlderDate(date1,date2:LONGINT):BOOLEAN;

{Compare two date/time numbers in LONGINT format: seconds since 1904.
 For dates in our era (1987), this number of seconds is greater than
 2^31, enough to make a Pascal LONGINT appear to be a negative number.
 Ideally, this function should do an unsigned compare, but MPW Pascal
 makes that difficult.  The trick used here is transform both dates
 with an unsigned divide-by-2 (BitSR) and compare the results.  This
 works fine except when the two dates are within one second.  That's
 good enough for me.}

BEGIN
    OlderDate := BitSR(date1,1) < BitSR(date2,1);
END;
```

```
{---------------------------  AvoidFile  ----------------------------}

FUNCTION AvoidFile(fName:Str255):BOOLEAN;

{Check to see if fName is a file we should not back up.}

    BEGIN
        AvoidFile :=
            EqualString(fName,'Backup',FALSE,FALSE) OR
            EqualString(fName,'Desktop',FALSE,FALSE) OR
            EqualString(fName,'Clipboard File',FALSE,FALSE) OR
            EqualString(fName,'MPW.Suspendstate',FALSE,FALSE) OR
            EqualString(fName,'System',FALSE,FALSE);
    END;

{---------------------------  UserAbort  ----------------------------}

FUNCTION UserAbort:BOOLEAN;

    {Return TRUE if Command-Period has been entered.}

    VAR
        theEvent    : EventRecord;
        theChar     : BYTE;

    BEGIN
        UserAbort := FALSE;
        bDummy := GetNextEvent(keyDownMask, theEvent);

        IF theEvent.what = keyDown THEN
            BEGIN
            theChar := theEvent.Message MOD 256;
            IF (theChar = PeriodKey) AND BitTest(theEvent.modifiers,cmdKeyBit) THEN
                UserAbort := TRUE;
            END;
    END;

{---------------------------  InitBackup  ---------------------------}

PROCEDURE InitBackup;

    {Clear statistics for the start of a backup.}

    BEGIN
        numFoldersScanned := 0;
```

```
        numFilesScanned := 0;
        numFilesCopied := 0;
        numDestDisks := 0;
    END;

{---------------------------    EjectDisk    ---------------------------}

PROCEDURE EjectDisk(drvNum:INTEGER);

    VAR
        volName      : Str32;
        vRefNum      : INTEGER;
        freeBytes    : LONGINT;
        again        : INTEGER;

    BEGIN
        FOR again := 1 TO 3 DO
            {Eject only floppy disks!}
            IF (drvNum = 1) or (drvNum = 2) THEN
                BEGIN
                oDummy := GetVInfo(drvNum, @volName, vRefNum, freeBytes);
                oDummy := UnMountVol(@volName,vRefNum);
                oDummy := Eject(@volName,vRefNum);
                END;
    END;

{-------------------------    BuildPathName    -------------------------}

PROCEDURE BuildPathName;

{Build the current path name from the master list of folder names in
  the pathList[] array.}

    VAR
        i        : INTEGER;

    BEGIN
        pathName := '';
        FOR i := 1 to pathLen DO
            pathName := CONCAT(pathName,pathList[i],':');
    END;
```

```
{--------------------------  DrvName  ---------------------------}

PROCEDURE DrvName(VAR str:Str255; drvNum:INTEGER);

    BEGIN
        CASE drvNum OF
            1 : str := 'Internal drive';
            2 : str := 'External drive';
            OTHERWISE str := '';
        END
    END;

{--------------------------  SetDialogText  ---------------------}

PROCEDURE SetDialogText(itemNum: INTEGER);

    {Set one of the several dialog Static Text items used to display
    Backup progress status:}

    VAR
        itemType   : INTEGER;
        itemHdl    : HANDLE;
        itemBox    : RECT;
        newStr     : Str255;

    BEGIN
        {Set newStr as required for item:}
        CASE itemNum OF
            SrcDrvText:  DrvName(newStr,srcDrvNum);
            DestDrvText:  DrvName(newStr,DestDrvNum);
            SrcVolText: newStr := srcVolName;
            DestVolText: newStr := destVolName;
            PathText: newStr := pathName;
            SrcNameText: newStr := srcFName;
            NumFoldersText: NumToString(LONGINT(numFoldersScanned), newStr);
            NumFilesText: NumToString(LONGINT(numFilesScanned), newStr);
            FreeBytesText: NumToString(LONGINT(destFreeBytes), newStr);
            NumFilesCopText: NumToString(LONGINT(numFilesCopied), newStr);
            NumDestDisksText: NumToString(LONGINT(numDestDisks), newStr);
        END;

        {Get handle to dialog item and store newStr as its new text:}
        GetDItem(theDialog, itemNum, itemType, itemHdl, itemBox);
        SetIText(itemHdl, newStr)
    END;
```

```
{------------------------  SetAllDialogTexts  ----------------------}

PROCEDURE SetAllDialogTexts;

    {Set all dialog status items:}

    BEGIN
        SetDialogText(SrcDrvText);
        SetDialogText(DestDrvText);
        SetDialogText(SrcVolText);
        SetDialogText(PathText);
        SetDialogText(SrcNameText);
        SetDialogText(NumFoldersText);
        SetDialogText(NumFilesText);
        SetDialogText(NumFilesCopText);
        SetDialogText(DestVolText);
        SetDialogText(FreeBytesText);
        SetDialogText(NumDestDisksText);
    END;


{--------------------------  GetSrcDestInfo  --------------------------}

PROCEDURE GetSrcDestInfo;

    {Get the volume name, reference number, and free space on the volumes
     currently mounted in the source and destination drives.}

    VAR
        err     : OSErr;

    BEGIN
        err := GetVInfo(srcDrvNum, @srcVolName, srcVRefNum, srcFreeBytes);
        if (err <> NoErr) THEN
            srcVolName := '(ejected)';

        err := GetVInfo(destDrvNum, @destVolName, destVRefNum, destFreeBytes);
        if (err <> NoErr) THEN
            BEGIN
            destVolName := '(ejected)';
            destFreeBytes := 800*1024;
            END;
```

```
        {Update display:}
        InitBackup;
        SetAllDialogTexts;
    END;

{-------------------------  CreateDestFolders  ------------------------}

PROCEDURE CreateDestFolders;

    {Create a set of nested folders using the folder names in the
     pathList[] array.  If any of the folders already exist,
     the PBDirCreate function leaves them alone.}

    VAR
        destCPB    : CInfoPBRec; {Record used by PBGetCatInfo}
        destPB     : HParamBlockRec; {Record used by PBDirCreate}
        destPBName : Str255; {Name of the folder being created}
        destPathName : Str255; {Path of folders created so far}
        destDirID : LONGINT; {Directory ID of destPathName}
        i              : INTEGER; {Index to pathList array}

    BEGIN
        destDirID := 0;          {Start at root ...}
        destPathName := destVolName; {... of dest volume}

        {For each folder name in pathList:}
        FOR i:=1 TO pathLen DO
            BEGIN
            WITH destPB DO    {Set up record for PBDirCreate:}
                BEGIN
                ioCompletion := NIL;
                ioFDirIndex := 0;
                ioNamePtr := @destPBName;
                ioVRefNum := destVRefNum;
                ioDirID := destDirID; {Set ID of folder in which to nest}
                END;                  {the new folder.}

            destPBName := pathList[i]; {Set name of new folder}
            {Create new folder, throwing away error code, which may well be
             -48, the "duplicate name" error.}
            oDummy := PBDirCreate(@destPB, FALSE);
```

```
            WITH destCPB DO    {Set up for PBGetCatInfo}
                BEGIN
                ioCompletion := NIL;
                ioFDirIndex := 0;
                ioNamePtr := @destPathName;
                ioVRefNum := destVRefNum;
                END;

            {Use PBGetCatInfo to find the DirID of the folder we just
             created and save it in DestDirID for the next level.}
            DestPathName := CONCAT(DestPathName,':',pathList[i]);
            oDummy := PBGetCatInfo(@destCPB, FALSE);
            DestDirID := destCPB.ioDirID;
            END;
    END;

{---------------------------  CopyFileSrcToDest  ----------------------}

FUNCTION CopyFileSrcToDest(fName:Str32): OSErr;

    {Copy a single file from src to dest. Copy both resource and data
     forks and then set the new dest file's "finder info" and dates of
     creation and last modification to same as src file's. Return
     NoErr if copy went OK, IOErr if not.}

    VAR
        copySrcName     : Str255;
        copyDestName    : Str255;
        copySrcPB       : ParamBlockRec;
        copyDestPB      : ParamBlockRec;
        copyStatus      : OSErr;

    {---------------------------  CopyFork  ----------------------------}

    FUNCTION CopyFork(FUNCTION OpenFork(fName: Str255;
                                        vRefNum: INTEGER;
                                        VAR fRefNum: INTEGER): OSErr): OSErr;

        {Copy either data or resource fork depending on whick "Open fork"
         function is passed in. Return NoErr if fork was copied OK, IOErr
         if not.}
```

```
    VAR
        srcRefNum, destRefNum: INTEGER;
        byteCount    : LONGINT;

    BEGIN {CopyFork}
        IF (OpenFork(copySrcName, 0, srcRefNum) = NoErr) AND
           (OpenFork(copyDestName, 0, destRefNum) = NoErr) THEN
            BEGIN
            REPEAT
                byteCount := BUFLEN;
                oDummy := FSRead(srcRefNum, byteCount, @copyBuf);
                oDummy := FSWrite(destRefNum, byteCount, @copyBuf);
            UNTIL (byteCount = 0);
            CopyFork := NoErr;
            END
        ELSE
            CopyFork := ioErr;
        oDummy := FSClose(srcRefNum);
        oDummy := FSClose(destRefNum);
    END;

BEGIN {CopyFileSrcToDest}

    {Build complete "volume:path:file" names for source and destination:}
    copySrcName    := CONCAT(srcVolName,':',pathName,fName);
    copyDestName := CONCAT(destVolName,':',pathName,fName);

    copyStatus := ioErr;

    {If dest file already exists, wipe it out:}
    oDummy := FSDelete(copyDestName, 0);

    {Set up for PBGetFInfo to get file's Creator and Type:}
    WITH copySrcPB DO
        BEGIN
        ioCompletion := NIL;
        ioNamePtr := @copySrcName;
        ioVRefNum := srcVRefNum;
        ioFDirIndex := 0;
        END;

    IF PBGetFInfo(@copySrcPB, FALSE) = NoErr THEN
        IF Create(copyDestName, 0, copySrcPB.ioFIFndrInfo.fdCreator,
                copySrcPB.ioFIFndrInfo.fdType) = NoErr THEN
```

```
            IF CopyFork(OpenRF) = NoErr THEN
                IF CopyFork(FSOpen) = NoErr THEN
                    BEGIN

                    {Have copied data and resource forks - now it's
                     time to set dest file's dates 'n' stuff to match
                      src file's:}
                    WITH copyDestPB DO
                        BEGIN
                        ioCompletion := NIL;
                        ioNamePtr := @copyDestName;
                        ioVRefNum := destVRefNum;
                        ioFDirIndex := 0;
                        END;
                    oDummy := PBGetFInfo(@copyDestPB, FALSE);
                        copyDestPB.ioFIFndrInfo := copySrcPB.ioFIFndrInfo;
                        copyDestPB.ioFICrDat := copySrcPB.ioFICrDat;
                        copyDestPB.ioFIMdDat := copySrcPB.ioFIMdDat;
                        oDummy := PBSetFInfo(@copyDestPB, FALSE);
                        copyStatus := NoErr;
                        END;

        {If copy failed part way through, delete dest file:}
        IF copyStatus <> NoErr THEN
            oDummy := FSDelete(copyDestName, 0);

        CopyFileSrcToDest := copyStatus;
    END;


{-------------------------    StopBackup    ----------------------------}

PROCEDURE StopBackup;

    {Show alert for "Backup terminated"}

    BEGIN
        destVolName := '';
        destFreeBytes := 0;
        iDummy := StopAlert(300, NIL);
        DrawDialog(theDialog);
    END;
```

```
{---------------------------  CDFilter  ------------------------}

FUNCTION CDFilter(theDialog: DialogPtr;
                  VAR theEvent: EventRecord;
                  VAR itemHit: INTEGER): BOOLEAN;

    {Filter routine for the "Change Disks" alert box.  It looks for a
     disk to be inserted and then makes sure it's formatted.}

    VAR
        temp      : BOOLEAN;
        tempPt    : POINT;
        drvStat   : INTEGER;
        err       : INTEGER;

    BEGIN
        CDFilter := FALSE;

        {Allow for disk-inserted events:}
        IF theEvent.what = nullEvent THEN
            temp := GetNextEvent(diskMask, theEvent);

        CASE theEvent.what OF
            diskEvt:
                BEGIN
                {A disk has been inserted! Get its drive number and status:}
                destDrvNum := LoWord(theEvent.message);
                drvStat := HiWord(theEvent.message);

                IF (drvStat = NoErr) THEN
                    BEGIN
                    {Disk is formatted OK, so return:}
                    newDestReady := TRUE;
                    CDFilter := TRUE;
                    END
                ELSE
                    BEGIN
                    {Disk was not OK so call system init routine to let
                     user choose if/how to initialize it:}
                    tempPt.v := 30;
                    tempPt.h := 30;
                    err := DIBadMount(tempPt, theEvent.message);
                    IF (err = NoErr) THEN
```

```
                    BEGIN
                    {Disk was initialized OK:}
                    newDestReady := TRUE;
                    CDFilter := TRUE;
                    END;
                END
            END;
        END;

    END;

{-------------------------  ChangeDestDisk  -------------------------}

FUNCTION ChangeDestDisk: OSErr;

    {Eject the current dest disk and activate an alert box to prompt the
    user for a new one.}

    VAR
        alertItem : INTEGER;

    BEGIN

        {If this is just a dry run, don't do anything with the disks.
        Just pretend a new 800K disk has been inserted:}
        IF dryRunFlag THEN
            BEGIN
            numDestDisks := numDestDisks + 1;
            SetDialogText(NumDestDisksText);
            ChangeDestDisk := NoErr;
            destFreeBytes := MaxFileSize;
            Exit(ChangeDestDisk);   { <------- Early exit!!! }
            END;

        {Eject disk and activate alert box:}
        EjectDisk(destDrvNum);

            newDestReady := FALSE;
            alertItem := NoteAlert(200, @CDFilter);
            SetCursor(GetCursor(watchCursor)^^);

            {Redraw our main screen - the alert wiped some stuff out:}
            DrawDialog(theDialog);
```

```
        IF (newDestReady) THEN
            BEGIN
            {New dest disk is ready, so get its volume name, etc.}
            oDummy := GetVInfo(destDrvNum, @destVolName, destVRefNum,
                               destFreeBytes);
            numDestDisks := numDestDisks + 1;
            SetAllDialogTexts;
            ChangeDestDisk := NoErr;
            END
        ELSE
            BEGIN
            {New disk not ready - user must have canceled:}
            StopBackup;
            ChangeDestDisk := ioErr;
            END
    END;

{-----------------------------    BackupDisk    ----------------------------}

PROCEDURE BackupDisk;

    {Traverse the entire disk looking for files newer than backupDate.
    If in dry run mode, just record the existence of these files.
    Otherwise, try to copy them to a folder on a floppy disk.}

    VAR
        err         : OSErr;

    {-------------------------    BackUpDir-------------------------}

    FUNCTION BackupDir(srcDirID: INTEGER): OSErr;

        {First scan all of the files in a given folder, then recursively
         scan all of its sub-folders.}

        VAR
            index     : INTEGER; {file/dir  directory  index}

        {-----------------------    BackupFile    ---------------------}

        FUNCTION BackupFile: BOOLEAN;

            {Backup a single file, changing disks first if necessary.}
```

```
VAR
    err             : OSErr;

BEGIN {BackupFile}
    {Compute length of file (plus a little padding):}
    srcFLen := srcCPB.ioFlPyLen + srcCPB.ioFlRPyLen +1024;

    {Display file's name and counter:}
    SetDialogText(SrcNameText);
    numFilesScanned := numFilesScanned + 1;
    SetDialogText(NumFilesText);

    {If file is older than current backup date (or if it's
     the Backup program itself), don't copy it.}
    IF OlderDate(srcCPB.ioFlMdDat,backupDate) OR
       AvoidFile(srcFName) THEN
        BEGIN
        BackupFile := TRUE;
        Exit(BackupFile); { <------- Early exit!!! }
        END;

    err := NoErr;
    IF (srcFLen > MaxFileSize) THEN
        BEGIN
        {File is too big to possibly fit on a disk - skip it!}
        ParamText(srcFName,pathName,'','');
        iDummy := StopAlert(500, NIL);
        DrawDialog(theDialog);
        err := NoErr;
        END
    ELSE
        BEGIN

        {Keep asking for new disks until one is inserted with
        enough free space for our file:}
        WHILE (srcFLen > destFreeBytes) AND (err = NoErr) DO
            err := ChangeDestDisk;

        IF err = NoErr THEN
            {big enough disk was inserted:}
            BEGIN
            IF dryRunFlag THEN
```

```
                    {Dry run only pretends to copy:}
                    destFreeBytes := destFreeBytes - srcFLen
                ELSE
                    BEGIN
                    CreateDestFolders;

                    {Try to copy the file:}
                    err := CopyFileSrcToDest(srcFname);

                    {Update destFreeBytes:}
                    oDummy := GetVInfo(destDrvNum, @destVolName,
                                       destVRefNum, destFreeBytes);
                    END;

                IF err = NoErr THEN
                    BEGIN
                    {File copied OK:}
                    numFilesCopied := numFilesCopied + 1;
                    END
                ELSE
                    BEGIN
                    {File not copied OK! Advise user and continue}
                    ParamText(srcFName,pathName,'','');
                    iDummy := StopAlert(400, NIL);
                    DrawDialog(theDialog);
                    err := NoErr;
                    END;

                SetDialogText(NumFilesCopText);
                SetDialogText(FreeBytesText);
                END;
            END;
        BackupFile := (err = NoErr);
    END;

BEGIN {---------- BackupDir --------}
    SetDialogText(PathText);
    numFoldersScanned := numFoldersScanned + 1;
    SetDialogText(NumFoldersText);

    {This REPEAT-UNTIL loop goes through all the files in this
    folder and calls BackupFile for each one:}
    index := 1;
```

```
REPEAT

    {Check for user's Command-Period key:}
    IF UserAbort THEN
        BEGIN
        StopBackup;
        BackupDir := IOErr;
        Exit(BackupDir); { <------ Early exit!! }
        END;

    srcFName := '';
    srcCPB.ioFDirIndex := index;
    srcCPB.ioDrDirID := srcDirID;

    { Get the info for the file }
    err := PBGetCatInfo(@srcCPB, FALSE);

    IF err = NoErr THEN
        IF NOT BitTst(@srcCPB.ioFlAttrib, 3) THEN
            IF NOT BackupFile THEN
                BEGIN
                {If BackupFile failed, quit:}
                BackupDir := IOErr;
                Exit(BackupDir); {<----- Early exit!! }
                END;
    index := index + 1;

UNTIL err <> NoErr;

{This REPEAT-UNTIL loop recursively calls BackupDir for each
sub-folder in this folder.}
index := 1;
REPEAT
    srcFName := '';
    srcCPB.ioFDirIndex := index;
    srcCPB.ioDrDirID := srcDirID;

    { Get the info for the file }
    err := PBGetCatInfo(@srcCPB, FALSE);

    IF err = NoErr THEN
        IF BitTst(@srcCPB.ioFlAttrib, 3) THEN
```

```
                BEGIN
                    pathLen := pathLen+1;
                    pathList[pathLen] := srcFName;
                    BuildPathName;

                    {Recursive call to BackupDir:}
                    IF BackupDir(srcCPB.ioDirID) <> NoErr THEN
                        BEGIN
                        {Exit if something went wrong:}
                        BackupDir := IOErr;
                        Exit(BackupDir); { <---- Early exit!! }
                        END;

                    err := NoErr;
                    pathLen := pathLen-1;
                    BuildPathName;
                    SetDialogText(PathText);
                    END;
            index := index + 1;
        UNTIL err <> NoErr;
        BackupDir := NoErr;
    END;

BEGIN {------ BackupDisk ------}
    GetSrcDestInfo;

    IF NOT dryRunFlag THEN
        {Make sure source and destination drives are different:}
        IF (srcDrvNum = DestDrvNum THEN
            BEGIN
            iDummy = StopAlert(600,NIL);
            Exit(BackupDisk);
            END
        ELSE
            IF (destDrvNum > 2) THEN
                IF (CautionAlert(700,NIL) <> 2) THEN
                    Exit(BackupDisk);

    IF (DestDrvNum <= 2) AND (NOT dryRunFlag) THEN
        IF ChangeDestDisk <> NoErr THEN Exit(BackupDisk);

    InitBackup;
    DrawDialog(theDialog);
```

```
        WITH srcCPB DO
            BEGIN
            ioCompletion := NIL;
            ioNamePtr := @srcFName;
            ioVRefNum := srcVRefNum;
            END;

        pathLen := 0;
        BuildPathName;

        {Backup entire disk:}
        oDummy := BackupDir(2);

        EjectDisk(destDrvNum);
    END;

{---------------------------  SetBackupDate  ----------------------------}

PROCEDURE SetBackupDate(itemNum:INTEGER);

    {Set the backup date/time variable:}

    VAR
        i : INTEGER;

    BEGIN
        GetDateTime(backupDate);
        CASE itemNum OF
            Hour1Cmd: backupDate := backupDate - 60 * 60 * 1;
            Hour8Cmd: backupDate :=  backupDate- 60 * 60 * 8;
            Hour24Cmd: backupDate := backupDate - 60 * 60 * 24;
            Hour48Cmd: backupDate := backupDate - 60 * 60 * 48;
            Week1Cmd: backupDate := backupDate - 60 * 60 * 24 * 7;
            Week2Cmd: backupDate := backupDate - 60 * 60 * 24 * 14;
            Month1Cmd: backupDate := backupDate - 60 * 60 * 24 * 30;
            Month2Cmd: backupDate := backupDate - 60 * 60 * 24 * 60;
            Year1Cmd: backupDate := backupDate - 60 * 60 * 24 * 365;
            WayBackCmd: backupDate := 0;
        END;

        {Check the appropriate menu item:}
        FOR i := 1 TO WayBackCmd DO
            CheckItem(myMenus[backupM], i, (i = itemNum));
        END;
```

```
{--------------------------  SetUpMenus  ----------------------------}

PROCEDURE SetUpMenus;

    {Get menu resources and display menu bar:}

    VAR
        i               : INTEGER;

    BEGIN
        myMenus[fileM] := GetMenu(fileID);
        myMenus[backupM] := GetMenu(backupID);

        FOR i := 1 TO menuCount DO InsertMenu(myMenus[i], 0);
        DrawMenuBar;

        SetBackupDate(Week1Cmd);
    END;

{--------------------------  DoCommand  ----------------------------}

PROCEDURE DoCommand(mResult: LONGINT);

    { Execute the manu command specified by mResult, the result of
      MenuSelect function. }

    VAR
        theItem     : INTEGER;
        theMenu     : INTEGER;
        temp        : INTEGER;
        templ       : LONGINT;

    BEGIN
        {Extract menu number and item number:}
        theItem := LoWord(mResult);
        theMenu := HiWord(mResult);

        CASE theMenu OF

            fileID:
```

```
            {User made a choice in the File menu:}
            CASE theItem OF
                BackupCmd:
                    BEGIN
                    SetCursor(GetCursor(watchCursor)^^);
                    dryRunFlag := FALSE;
                    BackupDisk;
                    SetCursor(arrow);
                    END;
                DryRunCmd:
                    BEGIN
                    SetCursor(GetCursor(watchCursor)^^);
                    dryRunFlag := TRUE;
                    BackupDisk;
                    SetCursor(arrow);
                    END;
                EjectCmd:
                    BEGIN
                    EjectDisk(1);
                    EjectDisk(2);
                    GetSrcDestInfo;
                    END;
                QuitCmd: doneFlag := TRUE;
            END;

        backupID:
            {User made choice from the Backup menu:}
            SetBackupDate(theItem);

    END;

    {Un-highlight the chosen menu and menu item:}
    HiliteMenu(0);
END;

{----------------------------  NextDrvNum  ----------------------------}

PROCEDURE NextDrvNum(VAR drvNum:INTEGER);

    {Set drvNum to the next drive number in the drive number list, taking
     care to detect the end of the list ane wrapping back to the front.}
```

```
    VAR
        i           : INTEGER;

    BEGIN
        i := 1;
        WHILE (i < numDrives-1) AND (drvNumList[i] <> drvNum) DO i:= i+1;
        IF (i = drvNum)
            THEN drvNum := drvNumList[i+1]
            ELSE drvNum := drvNumList[1];
        GetSrcDestInfo;
    END;

{----------------------------    BuildDrvNumList    ------------------------}

PROCEDURE BuildDrvNumList;

    {Scan the Macintosh drive queue, recording the drive numbers of all
     mounted volumes in the array drvNumList.  We'll also assume that
     both internal and external floppies are present.  This simplifies
     our logic and doesn't really hurt anything.  If a non-existent floppy
     is accessed, the Mac does not blow up.}

    VAR
        drvPtr     : QElemPtr; {Drive queue pointer}

    BEGIN
        {Assume both floppies, drives 1 and 2, are present:}
        numDrives := 2;
        drvNumList[1] := 1;
        drvNumList[2] := 2;

        {Run down the drive queue looking for other mounted disks.}
        drvPtr := GetDrvQHdr^.QHead;
        WHILE (drvPtr <> NIL) DO
            BEGIN
            IF (drvPtr^.drvQElem.dQDrive > 2) THEN
                BEGIN
                numDrives := numDrives+1;
                drvNumList[NumDrives] := drvPtr^.drvQElem.dQDrive;
                END;
            drvPtr := drvPtr^.drvQElem.qLink;
            END;
    END;
```

```
{------------------------------  Filter  ------------------------------}

FUNCTION Filter(theDialog: DialogPtr;
                VAR theEvent: EventRecord;
                VAR itemHit: INTEGER): BOOLEAN;

    {This is the filter routine for the Backup's main dialog box.  It
    does nothing but look for mouse-button-down events in the menu
    bar, in which case it activates the Toolbox's manu handler and then
    calls DoCommand to perform the menu action:}

    VAR
        temp      : BOOLEAN;
        theChar     : Byte;
        whichWindow     : WindowPtr;

    BEGIN
        Filter := FALSE;

        IF theEvent.what = nullEvent THEN
            temp := GetNextEvent(diskMask, theEvent);

        IF theEvent.what = mouseDown THEN
            IF FindWindow(theEvent.where, whichWindow) = inMenuBar THEN
                BEGIN
                DoCommand(MenuSelect(theEvent.where));
                Filter := TRUE;
                END;
    END;


{------------------------------  Backup  ------------------------------}

BEGIN
    { Initialize the system }
    {PLInitHeap(2500,NIL,FALSE,FALSE);   <---- unrecognized!?}
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);

    InitWindows;
    InitMenus;
    TEInit;
```

```
    InitDialogs(NIL);
    InitCursor;

    {Initialize main dialog and menu bar:}
    theDialog := GetNewDialog(100, NIL, POINTER( - 1));
    SetUpMenus;

    {Set source & dest drives:}
    BuildDrvNumList;
    destDrvNum := 1;
    srcDrvNum := 2;
    NextDrvNum(srcDrvNum);

    doneFlag := FALSE;
    REPEAT
        ModalDialog(@Filter, itemHit);
        {Handle the two buttons in the main dialog box:}
        CASE itemHit OF
            SrcDrvBtn   : NextDrvNum(srcDrvNum);
            DestDrvBtn  : NextDrvNum(destDrvNum);
        END;
    UNTIL doneFlag;
    DisposDialog(theDialog);
END.
```

Function OlderDate

OlderDate is a function called by BackupFile to compare the date of the current file to the BackupDate, the date chosen on the Backup last menu. If the file date is older than the backupdate, then the file does not need to be copied. Date/Time information consists of the number of seconds since 1904 and is stored in LONGINT format. Dates in our era (1987) are greater than 231; that is large enough to make a Pascal LONGINT appear to be a negative number. So what this function does is transform both dates with an unsigned divide-by-two (BitSR) and then compares the results. This trick works unless the two dates are within one second.

AvoidFile

AvoidFile contains a list of files that BACKUP can't copy. Ironically, you can't back up BACKUP. For some mysterious reason, trying to back up files currently in use causes a system error when you exit the program. This is not a serious problem as long as you have a copy of your System file and the BACKUP program somewhere on a floppy.

UserAbort

UserAbort is a Boolean function that calls the Event Manager to get the next keypresses. Specifically it's looking for Command-Period. If they have been pressed, it returns a value of true to the calling program.

InitBackup

This is a simple four-line procedure that clears the counting variables at the beginning of a backup.

EjectDisk

EjectDisk uses three toolbox routines: GetVInfo gets information about the mounted disks, UnMountVol, as expected, unmounts them, and the call to Eject, ejects the disks. EjectDisk is called by the procedure ChangeDestDisk whenever a disk is full and a new disk is needed, when Eject is chosen on the File menu, and when BackupDisk finishes the backup you've requested.

BuildPathName

BuildPathName takes the complete list of folders stored in the array pathList and puts them into the string variable pathName. It's called by BackupDisk and BackupDir.

DrvName

DrvName is a short procedure called by SetDialogText that takes the drive number and provides the information to add "Internal drive" or "External drive" when the screen is updated.

SetDialogText

SetDialogText is used by the following procedure SetAllDialogTexts when it needs to display or update backup status information on the screen. It gets the information, puts it into the format the dialog box requires, then stores it as a dialog item (GetDItem) and calls SetIText so when the Event Manager updates the dialog box, this new information will be included.

SetAllDialogTexts

SetAllDialogTexts goes through each field in the dialog box and, for each, calls SetDialogText to update it.

GetSrcDestInfo

GetSrcDestInfo uses the toolbox function GetVInfo twice, once for information about the source and again for information about the destination. If either is a floppy and there is no floppy inserted, it changes the volume name to "ejected" and calls SetAllDialogTexts to write this to the screen.

CreateDestFolders

CreateDestFolders creates the nested folders for the current file. It uses the toolbox function PPBDirCreate which only creates folders that do not already exist. It starts at the root of the destination directory,

and for each folder in the path sets up the record for PBDirCreate, puts the name of the new folder into the variable destPBName, and then calls PBDirCreate. If the function returns an error such as "duplicate name", we discard it because PBDirCreate does not create a duplicate folder. Then we set up information for another PB call, this one to PBGetCat Info to get the directory ID for the newly created folder. We save it in the variable DestDirID. We need it when we go through the loop again to create the next level of folders. We continue repeating this loop until we get to the innermost folder.

CopyFileSrcToDest

CopyFileSrcToDest first builds the complete pathnames and filenames for the source and destination files. Then it checks to see if there is already a file on the destination disk with exactly the same name as the file about to be copied. If there is, that file is deleted. Next, in order to copy a file, you must know its creator and type. So the next few lines set up pointers to this information. Then the toolbox routines PBGetInfo and Create are called to actually get the required information. Next CopyFork is called twice, first to copy the resource fork and a second time to copy the data fork. Then we set the information for the destination file so it matches that of the source file. (You probably never realized what the Finder had to do after you merely dragged a file icon from one disk window to another.) This includes copying all Finder information, the file creation date and the date of last modification. Toolbox functions that return values we don't need are put into the variable "dummy". If for any reason the copy failed part of the way through, we delete the file on the destination disk, and the last line gives CopyFileSrcToDest the CopyStatus to return to the calling procedure, BackupFile.

CopyFork

CopyFork is nested inside of CopyFileSrcToDest. It is called twice, once for the resource fork and again for the data fork. It first does a call

to open both the source file and the destination file. Then it sets the buffer length to 8K and calls FSRead to read the source file in 8K chunks. FSRead returns the actual number of bytes it read in ByteCount. Then FSWrite takes these bytes and writes them out to the destination file. Then the source and destination files are closed.

StopBackup

StopBackup deletes the filename and sets the available bytes value to zero. It uses the StopAlert resource and a call to DrawDialog to display the message "Backup terminated at your request". StopBackup is called by ChangeDestDisk if the user cancels or by BackupFile if UserAbort returns a value of true indicating the Command-Period keys were pressed.

CDFilter

CDFilter is the filter for the Change Disks alert box. It filters disk events. If a disk has been inserted and is formatted, it returns a value of true. If the disk is unformatted, it calls the system initialization routine to give the user the option to format.

ChangeDestDisk

This function is called by BackupFile each time it needs a new disk. It checks to see if this is a Dry Run. If it is and the destination is a floppy, it increments the number of floppies needed in the dialog box, sets the necessary flags and jumps out of the function. If it's a true backup and the destination is a floppy, it calls Procedure EjectDisk, and draws an alert box with the NoteAlert resource (see Figure 9-5).

NoteAlert is called with the address of CDFilter which returns a value

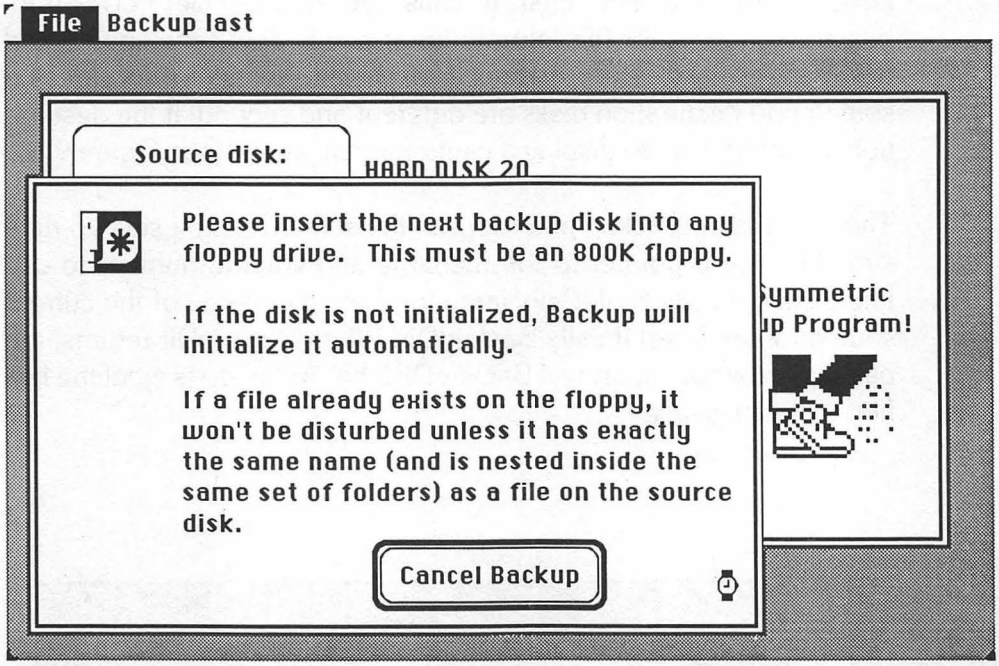


Figure 9-5. NoteAlert box.

of true when a disk has been inserted. After that ChangeDestDisk calls DrawDialog to redraw the main screen. Now we need to get information about the new disk with GetVInfo, increment the number of destination disks, and update the screen with this information. A call to SetAllDialogTexts takes care of this. If CDFilter returned a value of false, we're assuming the user pressed the Cancel buttons and we call StopBackup.

BackupDisk

BackupDisk is the heart of the program. Nested inside it are two procedures, BackupDir and BackupFile that do a good portion of its work.

Here's how it works. First it calls the routine GetSrcDestInfo, described above, for the information it needs about the source and destination disks. Then it does two checks — first to be sure the source and destination disks are different and second, if the destination is a hard disk, to display a cautionary message (see Figure 9-6).

Then it calls InitBackup and redraws the screen. Next it sets up data structures for a pointer to the filename and volume number to use later. Then it calls BuildPathName for the entire name of the current source folder. Then it calls BackupDir. When BackupDir returns, the backup is complete, and all BackupDisk has left to do is eject the last floppy that was used.

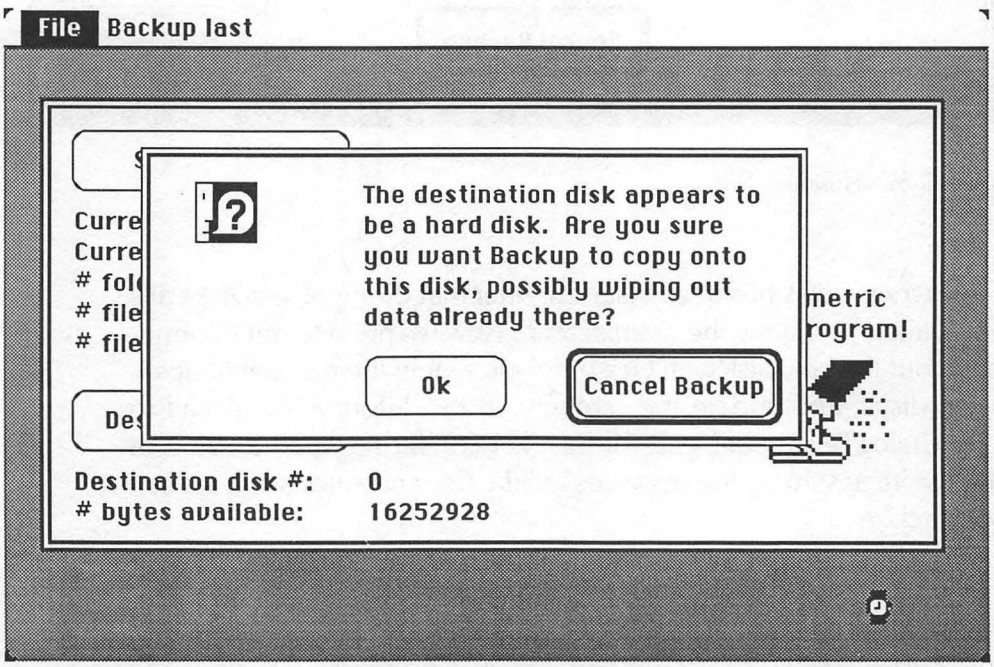


Figure 9-6. Alert box.

BackupDir

BackupDir has two major REPEAT-UNTIL loops. The first one checks UserAbort to see if Command-Period was pressed. If not it puts the information PBGetCatInfo needs into the srcCPB variables and then calls PBGetCatInfo to get information about the current file or folder. It returns no error if a file or folder was found (that is if we haven't fallen off the end of the directory list). Then it uses the line:

```
IF NOT BitTst(@srcCPB.ioFlAttrib,3)
```

to check the info it got to see if it was a folder or a file. If it is a file, it calls BackupFile, the procedure that calls CopyFileSrcToDest, to do the actual copying. This loop is repeated until it comes to the end of the list of files in the current folder.

The second REPEAT-UNTIL loop again sets up information for GetCatInfo and calls it. Then it looks to see if the information it got was a folder with this line:

```
IF BitTst(@srcCPB.ioFlAttrib,3)
```

Once we have a folder, we increment pathLen and put its name into the pathList array. Then we call BuildPathName to put the entire name into a string.

Next is a recursive call to BackupDir to traverse through the files in this folder. This continues until there are no more folders and files left in the folder. Then we strip off the folder name of the folder we've just looked at so we're back to where we started with this folder and can go on to another sister folder in the directory tree. Then we go through the REPEAT-UNTIL loop with the next folder.

BackupFile

BackupFile is called by the first REPEAT-UNTIL loop in BackupDir when it finds a file rather than a folder. It sets a variable srcFLen to the

length of the data fork plus the length of the resource fork plus a little padding (about 1K). Then it displays the name of the current file with the toolbox function SetDialogText, increments the number of files scanned, and displays this number, too, with the function SetDialog-Text. Next it checks the file date to see if it is one that meets the Backup last date criterion by calling OlderDate and checks to be sure it's a legitimate file to copy by calling AvoidFile. Next it checks the file size. If it's greater than 800K, that is if the file will not fit on a floppy disk, it displays the StopAlert message explaining that the file is too large and then continues on (see Figure 9-7). If the file is less than 800K, we keep asking for disks until we get one with enough free

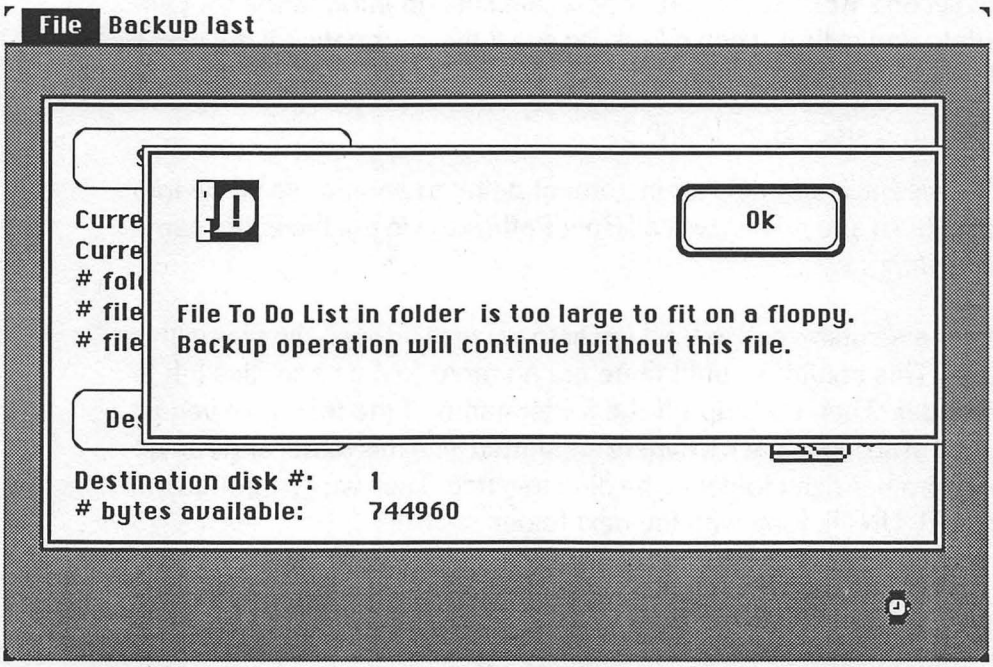


Figure 9-7. StopAlert message.

space for the file. We do this by comparing srcFLen to destFreeBytes. If srcFLen is greater, we call ChangeDestDisk to request a new disk.

Next we call CreateDestFolders to make the nested folders for the file (if needed). Then we call CopyFileSrcToDest with the source and destination file and folder names. Next is a call to GetVInfo one more time. It returns an updated value in the parameter destFreeBytes telling us how much available space is left on the floppy disk. Then if the file copy succeeded, we increment the number of files copied. If it failed, we use the StopAlert resource, notify the user that the copy for this file failed and continue on. Finally we update the information on the screen calling SetDialog Text with the number of files copied and the amount of free space.

SetBackupDate

SetBackupDate is called by DoCommand, when the user chooses a date from the Backup last menu. First it uses the built-in procedure GetTimeDate to look at today's date. Then it uses a case statement to compute the value of BackupDate based on today's date. As mentioned earlier, time/date information is stored in seconds, so we compute the number of seconds in the time period checked on the menu, subtract this value from today's date, and store this value in the variable BackupDate. Last we delete the check from any previously selected menu item and put a check mark in front of the current menu selection. The FOR loop with this toolbox call:

CheckItem(myMenus[backupM]i,(i = itemNum))

takes care of the entire process.

SetUpMenus

SetUpMenus does exactly that — it gets the menu resource for each menu, inserts the items on the menus, and draws the menu bar. Next

it calls the previous procedure SetBackupDate with the Week 1Cmd parameter to set the default backup date to one week.

DoCommand

DoCommand is called by the procedure that follows it, Filter. Filter picks out events that are menu selections and then calls DoCommand. DoCommand takes the parameter mResult and divides it into the LoWord which is the menu choice or result and the HiWord which is the menu name. Next is a case statement. First are the File menu choices for each command. Next is the only choice if a command on the Backup list menu was selected, that is a call to BackupDate with information about which item was selected. Last is a toolbox call that removes the highlight from the menu item chosen — HiliteMenu(0).

NextDrvNum

NextDrvNum is called by the main program to find the first hard disk in the queue. The two floppy drives are drive one and drive two. The hard disk(s) can be any number between 2 and 21. It takes the first hard disk it finds and makes it the proposed source disk.

BuildDrvNumList

BuildDrvNumList looks at the drive queue and builds the array drvNumList. It takes the shortcut of assuming both an internal and external floppy drive are present. The shortcut does not cause the program any problems.

Filter

Filter works like the Filter function in WHEREIS and LOCKIT. It checks with the Event Manager to get the events it wants. What we

want to filter out and use are mouse down events in the menu bar. If we find one of these, we call DoCommand and Filter returns a value of true.

BACKUP

The main program does the standard system initialization and calls GetNewDialog and SetUpMenus to take care of those initial tasks. Next we call BuildDrvNumList, set the default destination to the internal floppy drive, and call NextDrvNum to determine the default source, the hard disk. But the main work of the program is the REPEAT-UNTIL loop that calls ModalDialog, the system dialog box controller, with the address of Filter and the variable itemHit which takes care of the drive buttons. Filter filters out mouse down events in the menu area, calls DoCommand to take the correct action depending on the menu choice, and the program goes into action, responding to menu choices, and most likely doing either a Dry Run or an actual backup.

RESOURCES

Finally, here is the resource specification for BACKUP (see Figure 9-8). Because the list is so long, it's just listed in MPW format, but based on the previous samples in WHEREIS and LOCKIT, if you work with TML Pascal, you should be able to convert it. Note the additional menu resources. They give the program the information it needs to draw the menus.

BACKUP is a fairly long program and more difficult than the previous two programs. But if you've worked your way through the simple program, LOCKIT, and then the more complex WHEREIS, you should be able to follow the BACKUP program logic, learn a bit more about Mac programming, and end up with an amazingly useful backup utility.

Figure 9-8. MPW Resource list.

```
                                        [MPWResources.Backup]..


resource 'DLOG' (100, preload) {
        {55, 28, 275, 480},
        dBoxProc,
        visible,
        noGoAway,
        0x0,
        100,
        ""
};

resource 'DITL' (100, preload) {
        {       /* array DITLarray: 23 elements */
                /* [1] */
                {88, 280, 104, 432},
                StaticText {
                        disabled,
                        "An Amazing Symmetric"
                };
                /* [2] */
                {8, 8, 40, 152},
                Button {
                        enabled,
                        "Source disk:"
                };
                /* [3] */
                {24, 158, 40, 458},
                StaticText {
                        disabled,
                        "(src vol name)"
                };
                /* [4] */
                {48, 8, 64, 152},
                StaticText {
                        disabled,
                        "Current folder:"
```

```
};
/* [5] */
{48, 160, 64, 464},
StaticText {
        disabled,
        "(src folder name)"
};
/* [6] */
{64, 8, 80, 152},
StaticText {
        disabled,
        "Current file:"
};
/* [7] */
{64, 160, 80, 464},
StaticText {
        disabled,
        "(src file name)"
};
/* [8] */
{80, 8, 96, 152},
StaticText {
        disabled,
        "# folders scanned:"
};
/* [9] */
{80, 160, 96, 248},
StaticText {
        disabled,
        "(#folders)"
};
/* [10] */
{96, 8, 112, 152},
StaticText {
        disabled,
        "# files checked:"
};
/* [11] */
{96, 160, 112, 248},
StaticText {
        disabled,
        "(#files ch)"
};
```

```
/*  [12]  */
{112,  8,  128,  152},
StaticText  {
        disabled,
        "# files backed up:"
};
/*  [13]  */
{112,  160,  128,  248},
StaticText  {
        disabled,
        "(#files  cop)"
};
/*  [14]  */
{144,  8,  176,  152},
Button  {
        enabled,
        "Destination  disk:"
};
/*  [15]  */
{160,  160,  176,  344},
StaticText  {
        disabled,
        "(dest vol  name)"
};
/*  [16]  */
{200,  8,  216,  152},
StaticText  {
        disabled,
        "#  bytes  available:"
};
/*  [17]  */
{200,  160,  216,  248},
StaticText  {
        disabled,
        "(free  bytes)"
};
/*  [18]  */
{184,  8,  200,  152},
StaticText  {
        disabled,
        "Destination  disk  #:"
};
/*  [19]  */
```

```
                {184, 160, 200, 248},
                StaticText {
                        disabled,
                        "(#dest disks)"
                };
                /* [20] */
                {128, 368, 192, 432},
                Icon {
                        enabled,
                        11365
                };
                /* [21] */
                {9, 158, 25, 316},
                StaticText {
                        disabled,
                        "(src drive name)"
                };
                /* [22] */
                {144, 160, 160, 325},
                StaticText {
                        disabled,
                        "(dest drive name)"
                };
                /* [23] */
                {104, 264, 120, 448},
                StaticText {
                        disabled,
                        "Hard Disk Backup Program!"
                }
        }
};

resource 'DITL' (200, preload) {
        {       /* array DITLarray: 3 elements */
                /* [1] */
                {185, 174, 214, 288},
                Button {
                        enabled,
                        "Cancel Backup"
                };
                /* [2] */
                {8, 72, 91, 364},
                StaticText {
```

```
                        disabled,
                        "Please insert the next backup di"
                        "sk into any floppy drive.    This"
                        " must be an 800K floppy.\n\nIf the"
                        " disk is not initialized, Backup"
                        " will initialize it automaticall"
                        "y."
                };
                /* [3] */
                {101, 71, 181, 359},
                StaticText {
                        disabled,
                        "If a file already exists on the "
                        "floppy, it won't be disturbed un"
                        "less it has exactly the same nam"
                        "e (and is nested inside the same"
                        " set of folders) as a file on th"
                        "e source disk."
                }
        }
};

resource 'DITL' (300, preload) {
        {       /* array DITLarray: 2 elements */
                /* [1] */
                {34, 147, 63, 238},
                Button {
                        enabled,
                        "Ok"
                };
                /* [2] */
                {8, 68, 25, 309},
                StaticText {
                        disabled,
                        "Backup terminated at your reques"
                        "t."
                }
        }
};

resource 'DITL' (400, preload) {
        {       /* array DITLarray: 2 elements */
                /* [1] */
```

```
                {16, 273, 46, 353},
                Button {
                        enabled,
                        "Ok"

                };
                /* [2] */
                {72, 9, 132, 391},
                StaticText {
                        disabled,
                        "Could not copy file \"^0\" in fold"
                        "er \"^1\".  Backup operation will "
                        "continue without this file."
                }
        }
};

resource 'DITL' (500, preload) {
        {       /* array DITLarray: 2 elements */
                /* [1] */
                {16, 273, 46, 353},
                Button {
                        enabled,
                        "Ok"
                };
                /* [2] */
                {72, 9, 132, 391},
                StaticText {
                        disabled,
                        "File ^0 in folder ^1 is too larg"
                        "e to fit on a floppy. Backup ope"
                        "ration will continue without thi"
                        "s file."
                }
        }
};

resource 'DITL' (600, preload) {
        {       /* array DITLarray: 2 elements */
                /* [1] */
                {44, 178, 74, 258},
                Button {
                        enabled,
```

```
                "Ok"
            };
            /*  [2]  */
            {6,  58,  38,  330},
            StaticText  {
                disabled,
                "Destination disk drive is same a"
                "s source disk drive!  Cannot per"
                "form  backup."
            }
        }
};

resource 'DITL' (700, preload) {
        {       /*  array  DITLarray:  3  elements  */
            /*  [1]  */
            {101,  217,  133,  321},
            Button {
                enabled,
                "Cancel  Backup"
            };
            /*  [2]  */
            {101,  108,  132,  182},
            Button {
                enabled,
                "Ok"
            };
            /*  [3]  */
            {9,  106,  94,  320},
            StaticText  {
                disabled,
                "The destination disk appears to "
                "be a hard disk.  Are you sure yo"
                "u want Backup to copy onto this "
                "disk, possibly wiping out data a"
                "lready  there?"
            }
        }
};

resource 'ALRT' (200, preload) {
        {98,  20,  320,  385},
```

```
        200,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent
};

resource 'ALRT' (300, preload) {
        {36, 182, 106, 496},
        300,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent
};

resource 'ALRT' (400, preload) {
        {80, 80, 220, 480},
        400,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
```

```
        silent,
        OK,
        visible,
        silent
};

resource 'ALRT' (500, preload) {
        {80, 80, 220, 480},
        500,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent
};

resource 'ALRT' (600, preload) {
        {80, 80, 160, 410},
        600,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent
};

resource 'ALRT' (700, preload) {
        {80, 80, 220, 410},
        700,
```

```
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent,
        OK,
        visible,
        silent
};

resource 'MENU' (130, preload) {
        130,
        0xFFFF,
        enabled,
        "Backup last",
         {      /* array: 10 elements */
                /* [1] */
                "1 hour",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [2] */
                "8 hours",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [3] */
                "24 hours",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [4] */
                "48 hours",
                noIcon,
                noKey,
                noMark,
```

```
                plain;
                /*  [5]  */
                "1  week",
                noIcon,
                noKey,
                noMark,
                plain;
               /*  [6]  */
               "2  weeks",
               noIcon,
               noKey,
               noMark,
               plain;
               /*  [7]  */
               "1  month",
               noIcon,
               noKey,
               noMark,
               plain;
               /*  [8]  */
               "2  months",
               noIcon,
               noKey,
               noMark,
               plain;
               /*  [9]  */
               "1  year",
               noIcon,
               noKey,
               noMark,
               plain;
               /*  [10]  */
               "millenium",
               noIcon,
               noKey,
               noMark,
               plain
        }
};

resource 'MENU' (129, preload) {
        129,
        0x17,
```

```
        enabled,
        "File",
        {       /* array: 5 elements */
                /* [1] */
                "Backup",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [2] */
                "Dry run",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [3] */
                "Eject",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [4] */
                "-",
                noIcon,
                noKey,
                noMark,
                plain;
                /* [5] */
                "Quit",
                noIcon,
                noKey,
                noMark,
                plain
        }
};

resource 'ICON' (4854, preload) {
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
```

```
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
};

resource 'ICON' (11365, preload) {
        $"7FE0 7FF0 7FE0 7FF0 7FE0 FFE0 7FE1 FFE0"
        $"7FE3 FFC0 7FE7 FF80 7FEF FF00 7FFF FE00"
        $"7FC7 FC48 FF83 F800 8100 F832 80C0 2000"
        $"80F0 2000 810C 40B2 FE62 4000 4493 C000"
        $"4C92 02A2 5A66 0000 690B 0000 648B 8141"
        $"5A42 7000 4122 0800 4090 0450 4048 0400"
        $"7FE7 F400 4010 0800 3FFF F000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000"
};
```

As all programmers know, a program is never done. And BACKUP is no exception. A significant addition would be a way to select a folder or group of folders to backup. You could do this with a list box. The WHEREIS program has a model for building a list box. And, once again, you have an "exercise left for the reader . . . "

CHAPTER 10

Networks

THE CHANGING NATURE OF PERSONAL COMPUTERS

Personal computers were originally designed to be used by individuals. If you wanted to share information, you handed someone a floppy disk or sent a file using a modem and the telephone lines. You, not the DP department or manager, were in control of your machine and data. But as personal computers proliferated inside companies, the need to share information quickly and easily surfaced. Networks were seen as the answer. In some ways networks seem like a step back to central control of computer resources just as in the old DP days; in other ways networks are an absolute necessity.

Imagine this scenario. You're part of a company where several groups who work together — accounting, marketing, sales, manufacturing, and purchasing — need to share information on an on-going basis.

Each individual has his or her own files on his or her own computer and disks that no one else can get to. When you need something that someone else has, your only choice is to resort to inefficient and frustrating telephone tag. What you need is general access throughout the work group to the same documents, the ability to consolidate reports, circulate information for review and revision, and communicate efficiently without telephone tag.

Or imagine another scenario. Several people need information from the same database. All departments need current financial information to create budgets or make sales forecasts. Accounting and purchasing both need current inventory information. Requesting this information from the company mainframe is time-consuming. You have to wait for the DP department to print it, and then once you get it, you have to retype it into your Mac so you can include it in the spreadsheet or report you're doing. What you need is the ability to get the information you need when you need it and have it sent directly to your Mac.

Local Area Networks, or LANs as they are called, provide the ability to connect computers, printers, and hard disks so you can share information and resources. What you do is send a message addressed to one of the devices connected to the network. Devices that see the message is not addressed to them, ignore it. Computers on the network typically have equal access to devices, and this is what makes it possible to share disks, printers, and other devices.

Early network pioneers envisioned diskless computers hooked together sharing a large, central hard disk. They felt this would lower costs because individual computers wouldn't need their own drives. But if you consider the decreasing costs of individual hard disks compared to the costs of installing and wiring a shared hard disk, you may discover a shared hard disk will not justify the cost of a network. On the other hand, when several people need access to the same information, when you need quick turnaround, when you have costly laser printers

information. Typically you can also add print servers to share one or more printers and electronic mail servers to handle electronic mail.

Disk servers and file servers perform some similar functions, but like automobiles, a disk server contains the basics like a Volkswagen while a file server is a bit more elegant like perhaps a Volvo. The prime objective of a disk server is to make the disk available to everyone on the network. Typically it does this by partitioning the hard disk, the network. Typically it does this by partitioning the hard disk, and assigning passwords when you set up the network, and they are difficult to change later. If a partition is designated as private, it is typically limited to one user at a time. If it is public, multiple people can use it, but files are designated as read-only, and information in them cannot be changed. This is the only way the disk server can protect against more than one person writing to the same file at the same time and consequently destroying it.

File servers are more sophisticated. Most don't require proprietary partitioning. Several people can use the same volume, and the file server assigns write access to individual files on a first come first serve basis. With special multiuser software, file servers can incorporate record-locking. Several people can use the same file, even look at the same record, but only one person at a time can update it.

File servers can be dedicated or nondedicated. Dedicated file servers are used only to manage network traffic. A dedicated file server may be faster, is not susceptible to crashes on account of user errors, but is, of course, more expensive. Nondedicated file servers perform file server tasks but are individual workstations as well. This is a good arrangement if you need occasional use of the server as a workstation.

Print servers take print requests from network members, spool them to the server's disk, and send them to the proper printer when it is available. They are transparent to the user. Usually you simply use the application's Print command which the print server intercepts. It then

takes over and manages the printing. The machine that is the file server may also be the print server or the print server may be a separate machine.

An electronic mail server is a software extension to a file server for the purpose of managing electronic mail for people on the network. Inbox is a popular commercial electronic mail server. You can use it to route and manage phone messages and send notes and correspondence. You could, for example, send a contract to your cohort with suggestions for revisions. He or she could make the changes on screen and send it back with a quick note explaining the changes. You'd receive the revised contract and be ready to review and print it almost instantly.

Gateways

Gateways, or bridges in Apple's terminology, connect two or more networks. These can be similar networks, such as connecting two AppleTalk networks, or different networks, such as connecting an AppleTalk network to a PC network like Ethernet.

Centralized or Distributed

A centralized network has one server and all its resources attached to that server. Network users only have access to the disk and hardware resources of the server; resources of the stations are not directly available to network users. This makes controlling security on a centralized network easier. In a distributed network all the storage devices on one or more computers, if those computers are servers, are available to everyone on the network. This might mean that I could use parts of your hard disk and you could use my printer, and both of us could get current inventory information from the company database stored on another machine.

Now let's look at AppleTalk, the network that's built into the Macintosh, and then at four popular network servers that work with AppleTalk.

APPLETALK

If you were implementing an IBM PC network, you'd have a lot of difficult decisions. You'd need to decide on configuration and cabling. You'd need to look at network cards, operating systems, and software, and choose between dedicated or proprietary disk or file servers. And your cost would be the cost of the server and most likely at least $500.00 per machine connected to the network.

Implementing a network of Macs is simpler, mainly because of AppleTalk. AppleTalk is a relatively simple, low-cost bus network. You can buy cables and connectors to hook Macs together for $50.00 per machine. Cable is baseband, shielded twisted pair. The network connects to the AppleTalk port, which is essentially a serial port. And built into the Mac Plus's ROM is code to manage the network. Apple has made the configuration and cabling decisions for you and supplied the code that takes the place of a network card you'd plug into a PC. The server must, of course, be a Mac, so the only decision you need to make, once you decide to network, is what server software to use.

You can connect up to 32 devices to an AppleTalk network — Macs, printers, other computers, hard disks, and tape backup devices. You can use up to 1,000 feet of cable. The connectors at the Macintosh end of the cable consist of a small box with an isolation transformer to keep out electrical interference and at the other end a plug for the network cable (see Figure 10-4).

AppleTalk uses the bus topology. What this means is that there is no master controller. Each computer on the network is in charge of sending and receiving its own messages. To avoid collisions, the network

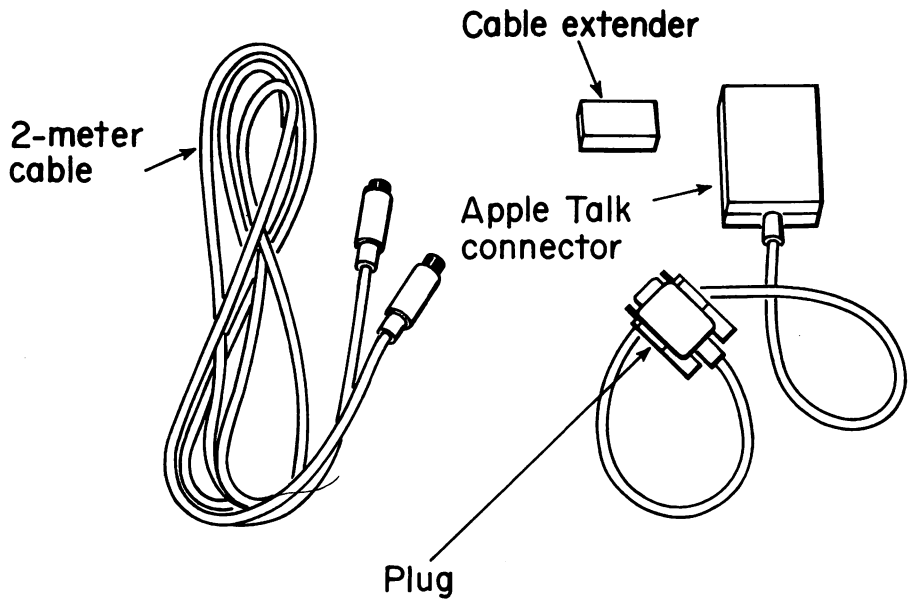


Figure 10-4. AppleTalk hardware.

uses CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol. This is the way it works. Any device currently using the network has priority. If you want to send something, you first listen to the network. If it is busy, you wait a randomly assigned amount of time and try again.

If the network is free, the sender first transmits a message to send which the receiver must acknowledge. If the receiver does not acknowledge the sender must free up the network and try again later. In addition to sending a message to a specific user, AppleTalk also allows you to broadcast a message, that is send it to everyone on the network.

Information travels on the network in packets or frames. Each frame consists of a synchronizing pulse and then header information containing the address of the recipient, the address of the sender, and information about the type of message. Next is the data. It can be up to 600K in length. If it is shorter, the synchronizing pulses indicate the length. Next is end-of-frame information. It consists of frame check information where both sender and receiver calculate an error correction value from information in the frame. If these match, all is well. If not, the message is discarded and the sender is notified to resend it.

If data is longer than 600K, AppleTalk presents a way to link frames together. But it only allows 12 frames to be sent at a time. That way no user can take over the network and prevent others from using it, too.

Most of the time, particularly for small messages, AppleTalk can work in the background. Your work won't be disturbed. If you receive longer messages, you may notice some disk activity that could temporarily interrupt your work.

APPLETALK NETWORK SERVERS

Four servers that use AppleTalk to network Macs are MacServe, TOPS, HyperNet and AppleShare. We'll look briefly at MacServe, TOPS, and HyperNet and then in chapter 11 take an in-depth look at AppleShare.

MacServe

MacServe is the only one of the four that is a disk rather than a file server. It operates in a very straightforward manner. What you do is partition your hard disk into up to 16 volumes. You specify a size for each of the volumes, and this size is fixed. It doesn't shrink or grow like HyperDrive drawers, and you can't resize it later like MacBottom partitions. So it's important to plan ahead.

When creating the volumes you designate them as private or shared. Private volumes are limited to one user at a time, and that user can both write to and read anything in the volume. When finished the user can release the volume so someone else can use it. Other volumes can be designated as shared and read-only. That way several people can use a file at the same time, but conflicts with more than one person writing to the file at the same time will be avoided. Most likely applications would be put in the shared volumes while data and document files that require updating would have to be kept in private volumes. But some applications that create temporary files must also be kept in private volumes. Applications in this category are Word, MacWrite, MacPaint, and PageMaker. They are designed for single users, and if they are shared and a second user starts the application, the first user's temporary file is erased and the system can crash.

Most of the servers work like this. You purchase the server software and install it on the server and on the other machines that will use information stored on the server's disk. The installation program adds the necessary software to the system folder and adds a network desk accessory. If you have a disk or printer that you also want to share that's attached to another machine in your network, that machine must also be a server and you'll have to buy an additional MacServe package for each additional server. Any machine on the network can get information from any other machine on the network that's designated as a server.

MacServe comes with a good print spooling program. It also has a volume manager with utilities to create the volumes, set up the print spooler, set up a disk cache to keep frequently used material in an easily accessible place, and assign passwords for each volume.

MacServe is a good server for connecting a small number of Macs, say two to four. The lack of flexibility of volume size and the limitation of one user per private volume, depending on the type of work you do, may make more than four users impractical. Also, if you want to share a disk that is already partitioned, for example a HyperDrive, unfortunately

you have to backup the disk, reformat it, and use MacServe's partitioning, and then copy your files back in. Currently Infosphere's MacServe only connects Macintoshes. It will not link PC's and and Macs.

TOPS

TOPS is a file server and then some. It's called an "interoperating system" because it can connect both Macs and PC's. Here's how it works. If a Mac is sending a disk or file command, TOPS intercepts the command, translates it into a TOPS command, then at the other end TOPS software on the target machine, say an IBM PC, translates it into a command the target machine can understand. It displays information on the target machine in that machine's format. On the Mac, for example, all files, even a Lotus 1-2-3 file, will have icons, while on a PC Mac files such as a Lotus Jazz document will be listed by name only in the standard MD-DOS directory format.

What this allows you to do is easily share information across different types of machines and let each machine process the kind of information it handles best. And you can do all this using the machine you prefer.

To install TOPS on a Mac you add the software and a TOPS desk accessory to the system files. You can use any partitioning system you currently have on your disk. On a PC you must plug in a board with an AppleTalk connector on the back. Then you run an install program. This copies in the TOPS utilities and adds a device driver to the PC's CONFIG.SYS file. Typically users won't know that TOPS is installed until they choose to share information with others on the network or need to use information from another computer.

TOPS is a true distributed network. With MacServe each network member with resources to share had to be designated a server, and some members could simply be stations — use resources of the server but not share their resources. On a TOPS network, resources of

each station can be made available to any other station. TOPS does this with a Publish function that looks much like the Apple's Font/DA Mover. Any user by publishing applications or documents can make them available to others on the network. And you can publish them in several ways. You can designate single-user application documents as read-only or one-writer to prevent corruption if more than one person tries to write at the same time. Or if you have special multi-user software such as Dbase III, you can designate it as many-writer. The application software performs record-locking. That way several people can use a file at the same time and many records can be updated, but only one person at a time can write to a particular record.

Because each machine can share its resources and use the resources of any other machine, each is a server on a TOPS network and consequently you must purchase a copy of the TOPS software for each machine you connect. The IBM package also includes an interface board. Both Macs and IBM's are connected with standard AppleTalk cables.

HyperNet

HyperNet is an excellent choice for a file server if you are going to share one or more HyperDrives. With HyperNet the servers must be HyperDrives. Individual users could, of course, download files from the server's HyperDrive to their own hard drives and these could be any brand drives, but these drives could not be shared.

HyperNet will be easy to learn to use for people who currently use HyperDrive. You get a file from a remote Mac much the same way you mount a HyperDrive drawer. You use a desk accessory to get a menu of available servers and drawers. Then you choose the server and the drawer you want to use. You'll then see an icon on your desktop and can use it just as you would if the drawer were on your own disk.

You can, of course, password protect drawers on servers just as you can drawers on individual HyperDrives. And HyperNet doesn't need to provide additional utilities for backup and print spooling because they've already provided them with HyperDrive. HyperNet comes with a Server menu that you can use to finetune performance for your specific needs. What you do is set a priority option — either giving the client (General Computer's substitute for user) or the server top priority, or choosing an option called Modified. Modified measures client and server requests and juggles between them in an attempt to be fair.

HyperNet avoids conflicts between users simultaneously writing to the same file by allowing multiple clients to share a file but giving only the first client write privileges. It also eliminates conflicts that can occur if several clients simultaneously use a single-user application that creates temporary files. HyperNet avoids collisions by giving a unique name to the temporary file of each application being used.

Wrapup

Networking with a small number of Macs, one of the servers described above, and AppleTalk is not difficult. It's easy to plug in the AppleTalk cables and install and use the server software. And as you might expect, all works in a very Mac-like way. Even adding a PC to a TOPS system is reasonably easy.

AppleTalk does have some speed limitations because the AppleTalk port is essentially a serial port. But there are some things you can do to make sharing information work more quickly. Ideally each machine on the network would have its own hard disk for local work. The server would only keep files that need to be shared and have extra storage perhaps for archiving. When you need to use a shared file you would get it from the server, copy it to your own machine, and then run it locally. This would mean you'd only need to make requests from the server at the beginning and end of a session rather than the continual requests you'd make if you ran the application from the server.

If you've added several new machines to your network and find performance is slower than you'd like, analyze work patterns. One network group found that they had basically two groups using resources — the sales group that frequently used the LaserWriter and the accounting group that made heavy use of the database. Since on AppleTalk only one computer and one device can talk at any given instant, accounting, for example, had to wait until sales finished sending instructions for the printer. A solution is to split these two groups into two separate networks, and then connect the networks with a bridge or gateway. That way when sales is using the printer, it doesn't slow down accounting's access to the hard disk. Each can have simultaneous access to what they need. Tying the two networks together gives both groups access to the hard disk and the LaserWriter, and both can be on the same electronic mail system. There will be delays only when one group wants to use the other's peripheral and it's already in use.

If you want to network on a grander scale, do your homework first. Very large networks linking several hundred Macs, huge file servers, and mainframes are possible. Several well-publicized examples are Martin Marietta and Seafirst Bank. Martin Marietta has a supernetwork linking 120 Macintoshes to a series of file servers with 200-400 MB hard disks and the entire system connected to a Sperry 11/92 mainframe. Seafirst Bank has linked over 2,000 Macs to 12,000 electronic devices in 190 offices throughout the state of Washington. In fact, there are plans for everyone at the bank with a telephone and a desk to have a Mac to get reports, send company-wide memos, or get the latest Treasury-bill rates.

While putting together a small-scale network is a simple task that most Mac users could accomplish, a network on a very large scale is not. Large networks composed of several interconnected networks connecting different types of PC's and peripherals require in-depth planning and technical people with extensive expertise in the specific type of network you envision.

CHAPTER 11

AppleShare

AppleShare is a file server for the AppleTalk network. It uses a Mac Plus for its server and can have up to nine hard disks attached to it — two Apple HD 20's and seven SCSI disks.

DEDICATED SERVER

The AppleShare server must be a dedicated machine. What this means is that its whole function is managing files; you can't also use it as a workstation. Having a dedicated server does require you to have an extra Mac, but typically networks with dedicated servers have more consistent performance because the server doesn't have to divide its time between network tasks and individual workstation tasks. Because of the dedicated server, Apple was able to provide a highly sophisticated set of privacy tools — tools that would not work if the server were also a workstation because then anyone using the

workstation would have access to everything on the server. Also, a dedicated server is less susceptible to human error and easier to maintain.

When your Mac is connected to an AppleShare file server, you'll see icons for the shared server disks on your desktop just as if they were directly attached to your machine. And you open the disk window, open folders, and choose files just as you do on your own disks. The Finder does change slightly when using information on the server. Private folders, ones that you don't have access to, are displayed but are gray and can't be opened. Folders that you own, ones that you've created and placed on the file server, are indicated with a black tab. Any folders that you can read but not write or change have a pen with a slash through it drawn in the information bar of the folder window.

One person on the network is designated as the network administrator. The network administrator is the only one that can look at the server information and perform server maintenance such as adding new user. He or she has a key that allows access to this information. The administrator signs up users for the server, assigns them passwords, and may put users into groups. Groups are handy if there are some folder to which that you want a select group of people to have access. The administrator is the only one other than the owner of a folder who can change access privileges. Typically, the network administrator is also in charge of backup up the server's hard disks.

ACCESS PRIVILEGES

You assign access privileges by folder. There are three kinds of access privileges. You can let people see folders nested inside a folder. You can let people see files and open them inside a folder. Or you can let people make changes to the content of a folder. You can assign any

combination of these privileges to the owner of the folder, members of a group, or to everyone on the network (see Figure 11-1).

If you wish, you can click the Enclosed folders also box and have these privileges apply to all folders nested inside of the current folder or you can assign privileges to each folder individually. The fact that you can assign privileges to a folder independent of all other folders is what makes AppleShare's access privileges outstanding. Most systems require you to designate an entire directory and all nested directories as private or shared. AppleShare's access privileges let you set up privileges that work the way you work.

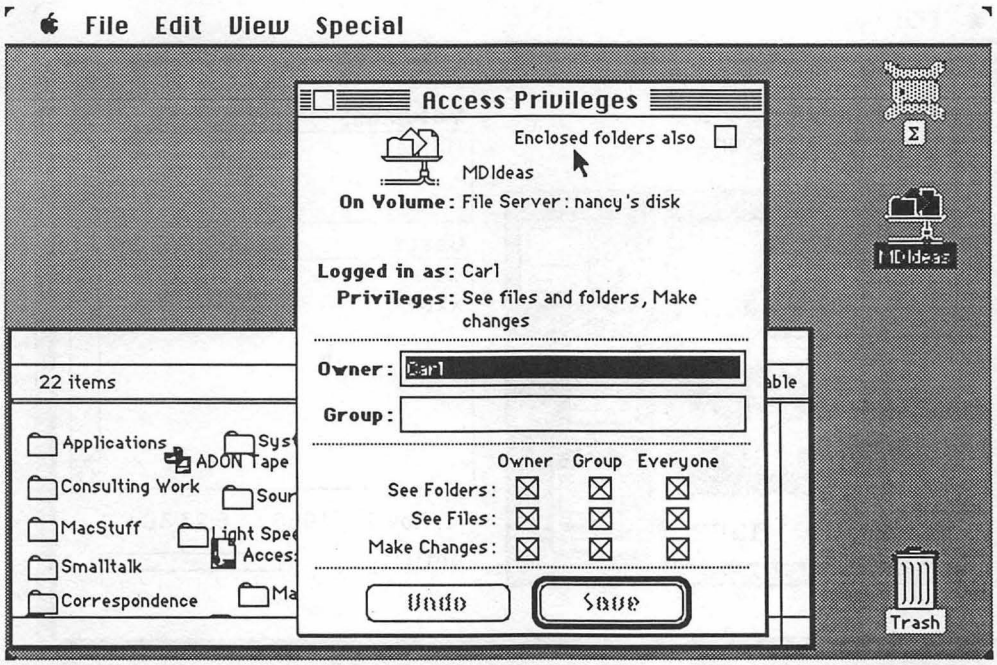


Figure 11-1. Access privileges.

SETTING UP

Installing AppleShare is fairly straightforward. We had the server set up and three Mac workstations connected and using the server with one hard disk attached in just over an hour. First you install the server by running the Admin program. You specify what disks will be volumes on the server, designate one as the startup volume, and the install program takes care of the details. Then you name the users and, if you choose, assign them to groups. When you finish and restart the system as the server, you will see a screen like the one in Figure 11-2.

It tells you which volumes or disks are available. In this example,

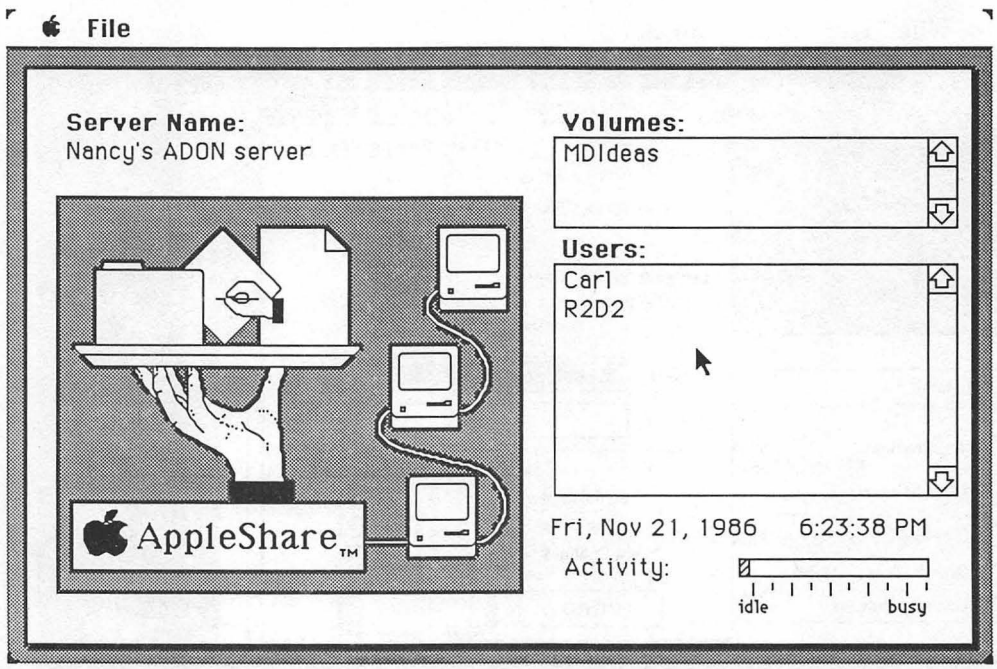


Figure 11-2. Server screen.

MDIdeas is the shared disk. It also lists which network users are currently logged on to the server. Then in the lower right corner is an activity meter. It changes position telling you how much the server is being used.

Once you've installed the server, the next task is to create startup disks for each workstation on the network. What you do is run the install program. Then you use the Control Panel to connect AppleTalk. Last you run the desk accessory Choozer (see Figure 11-3).

You specify your user name. This must match the user name the administrator assigned you. Click the AppleTalk Active button, check

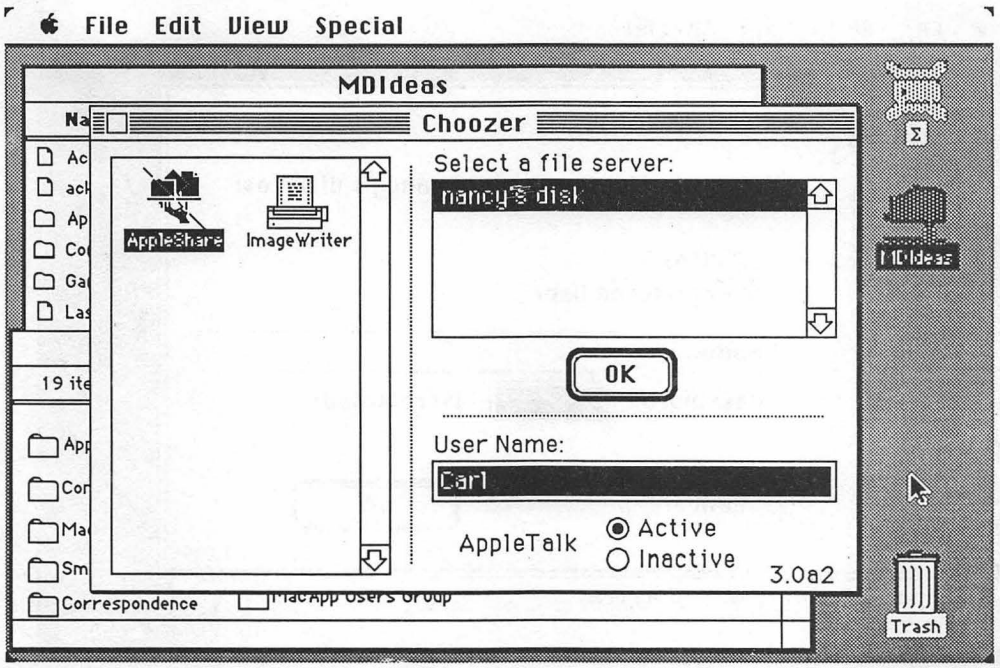


Figure 11-3. Choozer.

on the AppleShare button, select the name of the server or servers you want to use, and click OK. Choozer displays the dialog box shown in Figure 11-4).

Click the Registered User button, type your name and password and click OK. You'll see one more dialog box, this one asking you which volumes you wish to use and want opened each time you start your system. Select those by highlighting their names. You can select which volume you want open at startup by using the check boxes. Then you have one more choice. You can have the program save your name or your name and password. If you select name and password, you can eliminate entering your password each time you start your

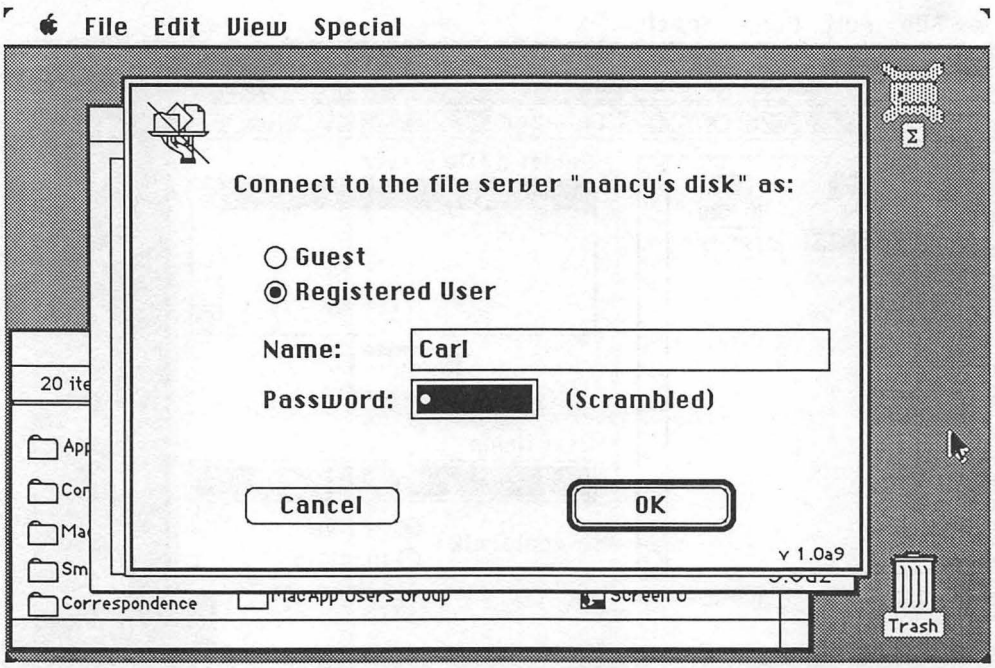


Figure 11-4. Connect to the server . . .

system. Now you're ready to go. Click OK and you'll see the server icon on your desktop and can use it just as you would your own disk.

MAINTENANCE

Occasionally the administrator will have to shut down the server for maintenance, for example, to add new users or assign groups or perhaps to back up all the information on the disk. When this happens, the administrator specifies the number of minutes until shutdown and each of the users is notified (see Figures 11-5 and 11-6).

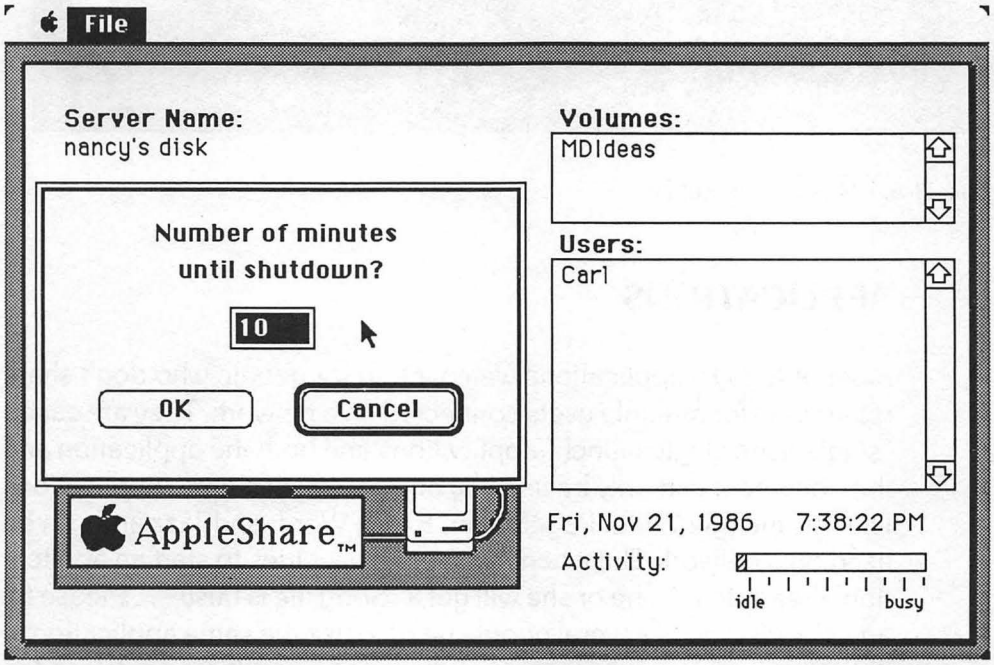


Figure 11-5. Server notice.

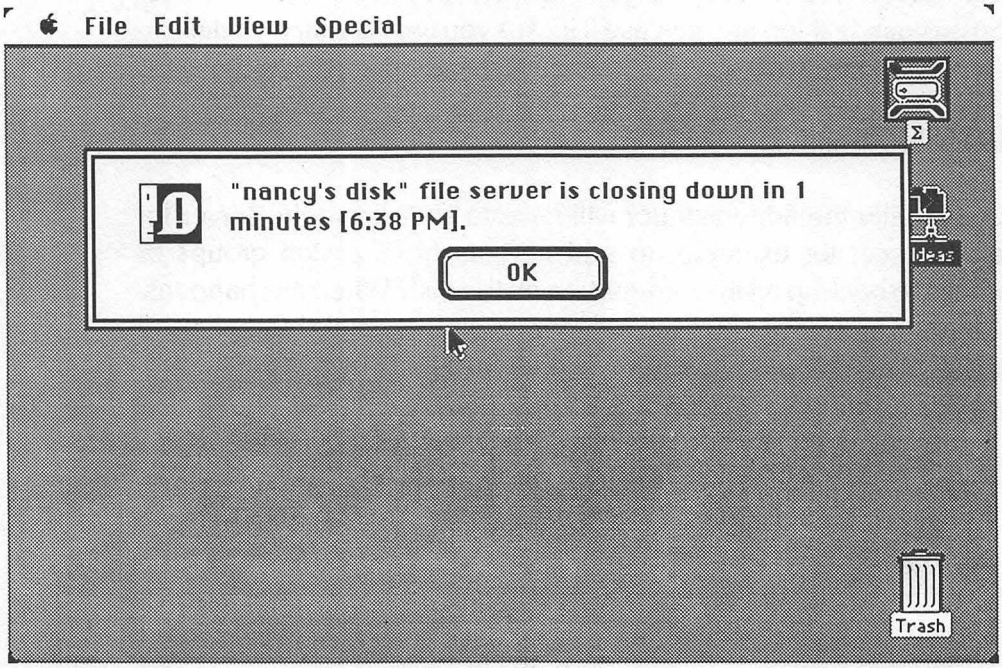


Figure 11-6. Workstation notice.

APPLICATIONS

Most of today's applications were written for people who don't share rather than for multiple users connected on a network. They are called "single user, single launch" applications and both the application and the document can only be used by one person at a time. Applications like this include Write, PageMaker, Excel, Word, and Jazz and can be used on a network. But when the second user tries to start an application already in use, he or she will get a "Sorry file is busy . . . Please try again" message. If several people need to use the same application at the same time on a regular basis, you can store multiple copies on the server in different folders.

Soon we'll see "single-user, multilaunch" as well as true multiuser software. Single-user, multi-launch applications allow several users to use the same application, but not the same document. True multiuser applications allow several users to use the same application and open the same document, but protect against two people simultaneously updating the file at the same time. At the writing of this book, there were no multilaunch or multiuser applications, but AppleShare has the tools available for developers to create true multilaunch or multi-user applications, and we should see a significant number of them soon.

Some of today's applications such as Draw, Word, Jazz, and Excel are accidentally multiuser. What this means is that it is possible for several people to edit the same document at the same time, but it is not safe. Some of these apps close the document file just after you load it, operate in memory. Because it appears that no file is open, the System allows a second user to open the document. But because these aren't truly multiuser, the application will not coordinate both people making changes simultaneously. What happened is the last person who saves overwrites the previous user's changes. (The server, however, will not be damaged.) There is a way to use even this type of app on AppleShare. What you need to do before you open the document is to drag it to a private folder and use it there. That way no one else can use it at the same time, and it will work safely.

Some applications that won't work correctly is stored on the server — applications that try to bypass standard Mac operating system call. However, you can still run these if you run the application from a local disk drive attached to your workstation instead of from a server volume. You can store your documents either at your workstation or on the server disk.

DESK ACCESSORIES

Each workstation can have its own desk accessories. And most desk accessories will function properly. But certain combinations of desk accessories or large numbers of them occasionally may cause a workstation some problems. Ones that are problematic are ones such as the Talking Moose that use the system heap. If you have problems with your desk accessories, remove all non standard ones. Then add them back one by one until you find the offender. Apple has included a list of "safe" desk accessories in the AppleShare Compatibility Guide available from dealers.

FOREGROUND/BACKGROUND APPLICATIONS

Even though AppleShare requires a dedicated server, you can run another network application such as an electronic mail server like InBox, a print server, a central database server, or even a multiuser calendar along with AppleShare. This gives you the flexibility and expandability to do more that just share applications and files on a hard disk.

FOR SALE

When you purchase AppleShare you get both administrator or server software and workstation software — two copies of each disk. You also get User's Guide's both for the administrator and for the workstation. If you need more workstation User's Guides, you can purchase a package of five (without software) from your dealer.

AppleShare is an impressive file server. It's easy to set up, comes with a sophisticated set of privacy tools, and because of the dedicated server, its performance is good, and it's straightforward to maintain. You can use single-user applications on AppleShare now, and because Apple has been working with over 100 third-party software developers, you'll soon have plenty of multiuser applications as well.

CHAPTER 12

What If . . .

DISK CRASH

If you've created the BACKUP utility in chapter 9 and use it or if you use the one that came with your hard disk, this chapter may be able to help you if you run into problems. It tells you what to do in the event of a disk crash. But this chapter won't be worth the paper it's printed on if you haven't backed up your data and so have nothing to restore.

The words "disk crash" sound dramatic and violent, and imply disastrous results. They remind one of airplane crashes or automobile crashes, and there are some similarities. Although with a disk crash you don't typically risk bodily injury, you can lose valuable data and with it a significant amount of time, incur mental anguish, and in certain situations, even lose your disk.

Although the results of a crash may be dramatic and violent, what

actually happens in a disk crash is not. Technically some of the time what happens is that a small particle of dust or even smoke bumps the read-write head, and the head "crashes" or hits the surface of the disk. This not only damages the data at the location of the head crash as you might expect but if the part of the disk that the head hits contains the directory or if it scratches a path across the entire disk surface, the entire disk may be unusable. Other times what happens is that the disk just won't start. This may be because a particle of dust destroyed the disk surface or directory, or it could simply be a mechanical malfunction. If there is no way to get the information on the hard disk, it is referred to as a crash, regardless of its cause. What happens in a situation like this is that you turn on your system but instead of booting from the hard disk, you keep getting the flashing disk icon with the question mark waiting for a System disk or you get a Mac with a sad face. It can't find the System on the hard disk. You try booting from the floppy that came with the hard disk, can start your Mac, but you don't get an icon for the hard disk. You obviously can't read from it or write to it; it's as if the hard disk weren't there.

I've been working with a number of Mac hard disks while writing this book. Two of them have crashed, but the disks weren't ruined and I was able to recover what I had backed up. And in working on my IBM I've experienced a number of hard disk crashes. Once in five years I did have to replace the disk. Once I had to reformat basically without warning. This meant I could only restore what I had previously backed up. Most of the time, probably once every two months, what happens is that it just won't start — I turn on my computer, it can't find the startup code on the hard disk, so it comes on with the BASIC code in ROM. On the PC it's a simple matter to take the cover off the machine, take a small screwdriver and move the stepper arm ever so slightly. On my machine this is typically enough to make the hard disk start up the next time I turn it on my computer. Unfortunately this kind of fix is not available for Mac hard disks. Manufacturers warn you not to remove the covers of the disks and opening the Mac invalidates its warranty.

The first time one of my Mac disks crashed I was working in Word. I got a dialog box with an unreadable, garbled message and an OK button. I clicked OK even though it was definitely not OK — I knew I was in trouble. The box wouldn't go away and it was coupled with a loud beep that wouldn't stop. I tried distracting it by attempting to choose commands from menus and pressing Command-Period, but the beep continued, the box wouldn't disappear, and nothing else worked. Finally I played my ace and turned off the Mac. When I tried to restart, I go the Mac with a sad face and a F0064 error message. This number means something happened to the System file on the disk and the startup code in ROM can't find the parts of the System it needs. I started from a floppy, put a new copy of the System on the hard disk, and was able to use it again. I was somewhat gun shy about just using the hard disk again without knowing what caused the System file to become unusable. I did after all lose the file I was working on when the disk crashed. So I ran the disk diagnositcs that come with HyperDrive. (See chapter 2.) These tell you whether you have bad sectors on the disk and need to reformat or that all is well. The peace of mind knowing the disk is really okay is definitely worth the five or so minutes it takes to run the diagnostics.

The second time one of the disks crashed I was again in Word, this time attempting to save a file. I got an alert box with the message: "Unrecoverable disk error on file: xxxxxxx". I clicked OK; again, it was NOT okay, The message kept coming back again and again, and I was never able to either save the file or quit Word. I flipped off the switch and then turned it back on. All seemed well. I loaded Word and the document I was writing. I tried to recoup — to get to the place I was when I got the nasty mesage. Unfortunately, when I tried to save, I got the "Unrecoverable disk error . . ." message again, and again had to turn off the machine.

When you face a situation like this, your disk may have crashed, but it isn't dead. There are two things to try; the message could come from either a bad System file or from a bad spot on the disk. First, hope it's

a bad System file. That's the easiest to fix. Do what I did after the first crash. Start from the floppy System software that came with your hard disk. Then drag a new copy of the System and Finder to your hard disk's icon. Then eject the floppy and choose Shut Down from the Special menu. This time you should start from the System on the hard disk and be ready to go. For peace of mind, if your hard disk came with diagnostic utilities, you may also wish to run them. If the disk checks out error free, all should be well. (But I'd run a backup program just for insurance.)

OTHER PROBLEMS AND FIXES

Bad Sectors

If even after copying in a new System, the unrecoverable disk error message persists, you may have a bad spot on the disk. Hard disks occasionally come from manufacturing with a bad sector or two. But the initial disk format picks up the bad sectors and tags them as unusable, and won't let applications or the Finder write to those sectors. But after a while, it is possible to develop an additional bad sector. Magnetic media is fragile. If this happens, you'll need to reformat the entire disk to pick up the bad sector and mark it as unusable. As with floppies, reformatting also erases the entire disk. So before you reformat, back up everything. If you're using the BACKUP program in chapter 9, choose the Milenium option. Then use your hard disk's utilities and reformat or initialize the disk. Then you can restore your applications and data and all should be well. This takes some time, but not as much time as repeatedly losing work when an application tries to write to a bad sector.

I had an experience with what apparently was a bad boot sector on the HyperDrive. When I turned on the Mac, it seemed to take forever to start. First I'd get the Mac icon with the smiley face. That would stay on the screen for two minutes. Then I'd get the HyperDrive logo for

another two minutes. Finally, I'd get to the desktop, and at that point all would seem fine until I had to restart. Eventually what I did was back up the entire disk. Then I reformatted the HyperDrive. This picks up the bad sectors and marks them as unusable. Then I restored my files and folders, and it start as quickly as it did when it was new.

Disappearing Folders

One person I know had what he called "creeping folder leprosy". One by one his folders became inaccessible. The Finder complained that not enough memory was available to open the folder and application's open boxes refused to open diseased folders. They neither said why nor displayed an error message. He then tried to rebuild the desktop file, the one with the directory (by holding down the Option key when starting the system), and got this message:

"Can't complete this operation because of disk errors"

The fix was to save as much as he could get to and then reformat — not ideal, but better than buying a new disk.

Software Error

It is possible to get a System error message with the number ID = 02. This is a software error message. Restart your system. Your folder or application file could be damaged. Try opening any folders nested in the current folder and check to be sure all your files are there. Then copy in another copy of the application you were using when you got this message.

System Error

If your System file on the hard disk is damaged, you may end up with a Finder desktop that looks strange or distorted or in the worst case

you may get System error messages displaying the not-so-friendly bomb. In a case like this what you need to do is to start from a floppy, the HD 20 Startup floppy or whatever startup disk came with your system. You force the floppy to be the startup disk by holding down the mouse button while your system starts. Then drag a copy of the System and Finder from the floppy to the hard disk. Be sure both the System and Finder are loose in the root directory or are in the same folder. They don't like to be separated.

If you have a HyperDrive and get a System error with the number ID = 15, you, too, have bad System software. You can try copying another System file to the startup drawer, but if this fails you'll have to follow the instructions in the HyperDrive documentation to recreate the startup drawer.

I use HyperDrive on a 512K Mac with the old ROMs and MFS. Some problems have been reported with the HyperDrive and the new ROMs which allow both MFS and HFS drawers. The system starts three or four times and the next time fails. A solution seems to be to make the startup drawer an MFS drawer. Other drawers can safely be either HFS or MFS.

Hard Disk Icon Is Not on the Desktop

You won't see the hard disk icon if you switched on your Mac without first switching on your hard disk. There is a possibility that your Mac will still start up if you have a floppy with a system folder on it in the internal drive. The fix for this one is obvious — Turn off your Mac. Turn on the hard disk. Then turn on your Mac.

USING THE DIAGNOSTICS

Most hard disks come with some type of diagnostic or testing program. Unlike initializing or formatting which completely erases the

disk, running a diagnostic test leaves the data on your disk intact. If you have a SCSI drive, the diagnostics create an error list. If the diagnostics find bad sectors not currently marked as bad, the test will fail and you'll have to format the disk. You can usually backup most of your work first. Then when you format, the formatting program reads this list and does not format the bad sectors.

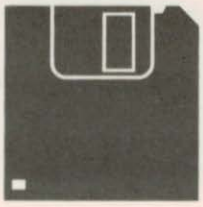
Both the Apple HD 20 and General Computer's Hyperdrive are pre-SCSI disks and their diagnostics work somewhat differently. Hyperdrive's diagnostics are part of the Hyperdrive Manager program. On the Manager's Test/Initialize menu, one item is "Run Disk Test". Choosing the disk test checks each sector on the disk and flags any bad sectors so they won't be used. If bad sectors are currently in use, the diagnostics may fail, and you'll have to reformat. However unlike SCSI drives, the format does not see the list of bad sectors. It simply portions out the disk in physical sectors. So after you reformat, you need to run the disk test a second time to mark the bad sectors as unusable.

Apple's HD 20 comes with a HD 20 Test application that works somewhat like Hyperdrive's. To run it you start your system from the HD 20 Startup floppy. When you see the message "Welcome to Macintosh", hold down the mouse button until the pointer turns into an arrow; this forces the floppy to be the startup disk. Then double-click on the HD 20 Test icon to start the test. Apple's test like General Computer's, checks each sector. If it fails, try reformatting and then run the test again to map the bad spots.

These tests can take as long as 15 minutes. At the end of the test, you'll get a message indicating whether your disk passed or failed. If the disk failed once, then you reformatted and ran the test again and it passed, you're in good shape. You can restore and get down to your real work. If it failed the second time, you'll have to contact your dealer or manufacturer for repair.

Index

Enclosed Is Your Utilities Disk...



This disk contains three indispensable ready-to-use utilities for your Macintosh hard disk. Simply place them inside your hard disk's System Folder and start enjoying the convenience and security that they provide. The three utilities contained are:

- **LOCKIT**—A utility to prevent unauthorized entry into your hard disk. Create your own special password that only you'll know. To activate **LOCKIT**, simply:

1. Place **LOCKIT** inside your hard disk's System Folder.

2. Activate the **LOCKIT** icon by placing the cursor on it and click once.

3. Pull down the **Special** menu and choose the **SetStartup** option. **LOCKIT** is now installed.

4. Once again go to the **Special** menu and choose the **ShutDown** option.

5. Restart your Macintosh from your hard disk. This time after the initial **Hello**, you'll be prompted to enter your password. Type the word **PassWord** and then press the **command** and **return** keys simultaneously.

6. You can now enter your own personalized password. **LOCKIT** is case sensitive, so be sure to remember if your personalized code is in upper case or lower case or a combination thereof. From here on you'll have to enter this password before you can use your hard disk.

7. Should there ever be a need for you to change your password, simply repeat steps 5 (but type in your current password) and 6.

- **WHEREIS**—A file locator that will locate your misplaced file(s) instantaneously. Activate it by double clicking it as you would any other utility. The screen will prompt you to type in the file name. This utility is not case sensitive, thus the file *MYFILE* can be located by typing *myfile*, *MyFile*, *mYfilE*, or any combination thereof.

- **BACKUP**—An indispensible utility for any Macintosh hard disk owner. This utility provides you with incremental true image backup. This utility will backup your hard disk to any other storage device; including: another hard disk, tape drive, or an 800K disk drive. A symmetrical backup utility program, **BACKUP** makes it possible for you to use any of the backups directly on your Macintosh in case of hard disk CRASH, thus giving you access to your Mac almost immediately, eliminating annoying down time. To use it simply double click its icon. Once the screen is on, you can do a **Dry run** first to make sure everything is in order, then you can choose to back up from any time increment from 1 hour to millenia.

SYSTEM REQUIREMENTS: Macintosh 512K Enhanced or Macintosh Plus with 800K drive and a hard disk drive.


ACKUP
FOR MACINTOSH 512K E AND PLUS
BANTAM ELECTRONIC PUBLISHING
0-553-50048-1
COPYRIGHT © 1987 BANTAM BOOKS

Managing Your Macintosh Hard Disk

Hard Disk Management for the Macintosh explains and gives solutions to some of the problems inherent with the use of a hard disk drive on the Macintosh; problems such as lost files, unauthorized use of files, and hard disk crash. These problems and the answers to them are of paramount importance to every hard disk drive user. This book also covers **AppleShare**—Apple's new file server, whcih is sure to become the standard in the networked Macintosh environment.

Everything you need in order to optimize your hard disk drive organizational and managerial skills are clearly defined in **Hard Disk Management for the Macintosh,** including:

- How to Manage Your Files—Properly organized filing systems minimize your chance of ever misplacing or even losing your file altogether.

- How to Backup Your Hard Disk—If misplacing or losing a file is not bad enough, how do you feel about losing every file in your hard disk drive through a CRASH? While preventing a crash is important, it is equally important to know what to do in case of a crash.

- How to Prevent Unauthorized Use—If your don't want anyone else to have access to your sensitive, personal or business data or communication, consider installing a file protection scheme as a safeguard.

- How to Use Your Hard Disk Drive as a File Server—Networking is a topic which is becoming increasingly important now that more and more Macintoshes are getting into the office environment. This book shows you how to effectively use the hard disk in a shared environment.

- How to Get the Most Out of AppleTalk and AppleShare—Two new products from Apple, which are destined to become the standard hardware and software, respectively, in any network involving the Macintosh.

The book also provides Pascal program listings, complete with resources for MPW and TML Pascals, for: LOCKIT—a passwork protection, WHEREIS—a file locator, and BACKUP—a utility program which enables you to backup to either a tape drive, another hard disk drive, or an 800K disk drive.

Nancy Andrews is author of *Microsoft File: Organizing Your Business on the Apple Macintosh* and *Windows: The Official Guide to Macintosh Operating Environment*. She is also the founder of Plain English, a company which specializes in writing software documentation for companies such as Microsoft, Maynard, and LifeWare. She is a frequent contributor to *PC Magazine* and *PC World*.

N 0-553-34398-X>3495

34398-X ■ IN U.S. $34.95 (IN CANADA $42.95) ■ BANTAM ELECTRONIC PUBLISHING