Mastering OS-9 on the Tandy Color Computer 3

An enjoyable, hands-on guide to OS-9 Level 2 complete with step-by-step tutorials. Requires an 80 column display (RGB preferred), two 40 track double sided disk drives, and 512K of memory.

Third Edition published in 1995 by FARNA Systems with permission

For questions and suggestions, please contact:

FARNA Systems
Box 321
Warner Robins, GA 31099-0321
or
FARNA Systems
c/o F. R. Swygert, Sr.
Rt. 4 Box 145
Leesville, SC 29070

Acknowledgements:
The people snd entities listed below all had a direct or indirect influence on the writing of this edition:
Microware Systems Corp.
Charles Askins
Andy Ball
Bill Brady
James Gibbons
Steve Goldberg
Marty Goodman
Michael Litt
Eric Miller
Dale Puckett
Rick Ulland
Bruce Warner
Richard White
And I have to personally thank Paul Ward, who gave FARNA Systems permission to edit and print the third edition (original title of first and second editions: "Start OS-9").

Foreword: Francis G. Swygert

This is my second major work concerning the Color Computer. It goes without saying that the majority of the work here is Paul Ward's, but editing and revising is still a lot ofwork!

It is not easy writing for this market, mainly because it is shrinking. There are few new CoCo users, and few purchasers of new material. Nevertheless, I still support this little beast, and a beast it can be under OS-9. I say this not only because of the difficulty some have with learning OS-9, but more because of the POWER that OS-9 unleashes. With this power comes a more complex way of doing things... very complex when compared to Tandy's simple Disk Extended Color BASIC.

I have decided to continue to support the CoCo, and will do so for many years to come. I love my little simple and powerful computer. Simple under DECB, powerful under OS-9. It works well for me, and I expect it to continue doing so for many years to come.

Preface: Paul K Ward

You are holding the only hands-on introduction to OS-9 Level 2. It differs from preceding books on OS-9 Level 2 because it is written for the intelligent beginner in a no-nonsense format, with clear, short examples which will speed you on your way to both USING and UNDERSTANDING OS-9 Level 2.

In "Mastering OS-9", we teach you the most-used features by providing sensible examples you perform yourself. This is our secret. "Mastering OS-9" brings the power of OS-9 Level 2 to your fingertips by putting those fingertips on the computer keyboard.

In hands-on tutorials you learn the most practical aspects first, one step at a time. By the end, you'll be able to explore with confidence aspects of OS-9 not covered here.

"Mastering OS-9" offers:
* ten tutorials, each providing immediate use of many powerful features of OS-9
* a wealth of clearly-written explanatory material to help you understand and retain what you learn
* appendices offering advice on software and hardware
* suggestions on customizing your system for special tasks

Here are some benefits you reap from finishing the tutorials:
* The investment in your computer system and OS-9 Level 2 wlll pay off.
* Your OS-9 computer will adapt to your work style.
* You will ultimately spend less time at your computer and put out a better product.
* You get a foundation in OS-9 that eases learning more advanced techn iques.
* You can make better-informcd purchases when buying hardware and software for your computer system.

OS-9 Level 2 offers broader benefits, too. People who aspire to be software developers will revel in OS-9 Level 2's programming environment, the most sophisticated you can find at most any price. Also, using business software available for your OS-9 computer puts professional power in your home and workplace for a modest investment. And OS-9 Level 2 is just plain fun. Please enjoy yourself.

CONTENTS

In the Beginning

Welcome To OS-9 Level 2

From the start, computers have been crotchety and demanding. The first computers took up large rooms and were expensive to maintain. It took years of training to design and run them. Large staffs of scientists swarmed over the equipment, constantly replacing tubes and adding new data.

When computers, now the size of a phone book, showed up in the American living room, they weren't much better. If you bought one, you understand. There were few programs on the market, so you often had to write your own. This required days, even weeks, of poring over confusing manuals to accomplish the simplest task. The few programs available were either games or were so puny and powerless it is a wonder we didn't all throw up our hands in frustration and call off the Computer Revolution. So why are computers still around? Blame VisiCalc.

VisiCalc was the first truly USEFUL program for a microcomputer. It excited business people because it solved complex financial calculations. It taught us that computers can help us plan better, thus saving our time and money.

Certainly VisiCalc for the Apple II computer convinced Apple to continue producing computers. It probably urged IBM into the home computer market. Other major names entered the market around this time, including Tandy/Radio Shack, Osborne, Timex, and others. Competition suddenly heated up in the personal computer industry.

In the end, the victors in this market struggle survived by following these principles:

* A computer won't sell unless it has great software.
* Great software is powerful and easy-to-use.
* Both the computer and the software need a large base of users to remain vital.
* Both the computer and the software need continued support from the manufacturers.

OS-9 Level 2 on the Color Computer 3 is a mature system which scores high marks on each of these tests — except one. OS-9 has a reputation for being less than easy to use.

A little work with a lot of reward

If you started out with Tandy's Disk Extended Color Basic or a variant (the CoCo's built-in software), OS-9 will require some rethinking. Be prepared. Mastering OS-9 is more difficult. But here's the payoff: your OS-9 based computer is harder to learn

because it off ers more than any other home computer system. It offers so much more that comparing the OS-9 “learning curve” with that of Disk Extended Color Basic is unfair.

OS-9 has a greater similarity to UNIX, the operating system found on larger, more expensive computers. Although UNIX also has a reputation for being confusing, its power and versatility have created a revolution in the computer industry. Most computer companies are scrambling to bring UNlX and UNIX-style operating systems to personal computers. UNIX “is driving many of our decisions at IBM,” according to IBM fellow Andrew Heller. IBM even moved its Scientific Division to the Entry Systems Division specifically “to accelerate the porting of IBM’s Unix operating system ... to the PS/2 Model 80,” one of IBM’s personal computers (BYTE Magazine, November 1987, p. 12.).

OS-9, smaller and faster than UNIX, brings a UNIX “feel” into your home. OS-9’s similarity to UNIX means you already have the power most other computer owners will have to wait for.

Some beneficial aspects of OS-9 Level 2 are listed below:
* It’s expandable. OS-9 Level 2 is a collection of over one hundred little programs, each of which does only one very small job. By adding new little programs, called utilities, or modules, OS-9 Level 2 can expand its power. New utilities and modules are included with “Mastering OS-9”. Your operating system has already grown!

* It’s Multi-tasking. OS-9 allows you to do several jobs at once. For example, you may wish to print Section One of an important report while you continue writing Section Two. The printing is the first job or “process”, and the writing is the second job or “process”. You can even check spelling on another document by creating a third process. OS-9 executes these three processes apparently at the same time.

* It has windows. To provide you visual access to several programs at once, Level 2 sets up windows for each. You have control over the function and appearance of each window. Choose colors, sizes, and whether the window must be able to display pictures.

Once each program has its own window, you select a window by pressing the CLEAR key. Each window appears on your screen in turn as you press the key. Need to write some text? Press CLEAR to get to your word-processor. Need to make some calculations? Press CLEAR to get to your spreadsheet. Each of these programs runs as though it has its own independent terminal.

* It helps you organize your work. Organized people have their files in file folders. OS-9 offers "files" of an electronic variety. You can group similar files in file directories (or simply directories). You can even group directories in other directories! This organizing principle, the hierarchical directory system, allows you to assemble your work neatly and sensibly.

* Automate dreary tasks.When you tell your computer to do something for you, you generally use commands. Sometimes, typing in commands is repetitious and time-consuming. OS-9 gives you an option. It doesn't care if its commands come from you or from a file.

If you know in advance all the commands you will need, just create a file which contains those commands, in order, to get your job done. What you are creating is a procedure file, so-called because it contains a list of procedures which OS-9 executes one at a time, without your intervention if you so choose.

In addition,
* OS-9 Level 2 is well-supported by the OS-9 Users Group and many enthusiasts worldwide.
* OS-9 has a strong future. Anything you learn about level 2 is applicable to OS-9/68000. Microware's client list reads like a Who's Who in industry. OS-9 is everywhere. It's the heart of the latest Compact Disk technology, CD-i, and the Interactive Television set-top boxes; computers from Japan to England to Tennessee, many of them state-of-the-art, run OS-9; and it is available on many different computers (including the Atari ST, Amiga, and even IBM clones). Your purchase of OS-9 Level 2 was a smart idea.

Why use OS-9 ?

Introduction

You have a job to do. OS-9 Level 2 provides the tools. In fact, it offers so many tools that becoming familiar with all 50 available commands is a big challenge.

Now for the good news: you will only use about 20 with any frequency. In "Mastering OS-9" we explain the most-used commands first so you can get right to work. Plus, you gain confidence in using OS-9 Level 2 which is sure to inspire you to experiment with the 30 or so commands we will not cover in depth.

Since it's best to learn by doing, "Mastering OS-9" teaches you what you need to know by having you perform simple exercises — each of which is extremely practical. You also learn:

* the “lingo” of OS-9. You discover explanations for new words in the main text, in end notes, and in essays. You learn how the words refer to your task at hand while avoiding words best left to the theorists.

* some OS-9 “theory”. A little theory is important to know. It provides a framework for understanding. Consequently, you will more likely remember what you are learning.

* how to build your own tools. OS-9’s procedure f iles help you create a system customized for exactly your needs.

How to use this book.

Pay close attention to the sub-headings in these margins. They will serve well as index tabs, as all items in them are indexed. All commands and important terms will be in bold print.

You should already be familiar with your CoCo 3 system. If you know these terms and how to use what they describe, you should be in good shape:

computer monitor
disk drive
printer
Multi-Pak Interface
reset b u tto n
write-protect tab
backup copies

If you don’t understand these terms, please take a while to become familiar with them by referring to the manuals provided with your CoCo 3 or a third party reference such as FARNA System's “Tandy's Little Wonder”. The OS-9 Level 2 manual is essential! If you do not have one, try to obtain a copy. Once you are comfortable with these terms, make a commitment to yourself to finish “Mastering OS-9”. Then, glance through it once or twice to get a feel for the structure of it. You should discover the following flow:

Introductory material — where you become familiar with the general terms, concepts, and commands involved in normal OS-9 work situations.

Getting Started with Tutorial *x* — where you become familiar with specific terms, concepts and commands mentioned in the upcoming tutorial .
Tutorial *x* — where you go through specific examples.
End-notes (if any) — comments on the tutorial.

Once you’ve leafed through this book, set up a loose schedule in your mind. You probably noticed that the hook’s contents are

divided into days (Day One, Day Two, etc.). At about ninety minutes a day, you would finish the book in about two weeks. Our suggestion, however, is to choose your own pace. It's best to go straight through the book, no skipping. The comments in the end-notes are designed to be thought-provoking and are usually less central to the tutorial.

At the end of the first "week" you'll find some essays. Read through them! You should find them interesting, light reading — perfect for the "Mastering OS-9" weekend. You'll find similar articles at the end of the second week, as well as valuable appendices on OS-9 topics, including the OS-9 User's Group.

Typographic Notations

Before we proceed we should explain the typographic notation used in "Mastering OS-9" to represent what your screen displays and what you must type. When you see

OS9:

in this book, it refers to the "OS9:" prompt printed to your screen by OS-9 (by the Shell, to be precise). DOn't type those characters yourself.You also see:

E:

in this book. This is the OS-9 Editor's prompt. DOn't type those characters either. Last, when you see:

<ENTER> or **<SPACE>**

press the key on your keyboard marked "ENTER" or the spacebar respectively. Other special keys will also be bracketed by the less than (<) and greater than (>) signs.

Other words and symbols are usually to be typed by you. For example, if we are discussing the dir (directory) command, we might print in the book:

OS9: dir <ENTER>

This indicates that, when you see the "OS9:" prompt, you should type "d", "i", and "r", followed by pressing the key marked "ENTER". Similarly, when you work with the OS-9 Editor, you might see in the book:

E: <SPACE> ***Type this text*** **<ENTER>**

Once again, don't type "E:" since that already appears on your screen. Go ahead and press the spacebar, type the text "Type this text" and then press the key marked "ENTER".

One last note about the disk which is included with "Mastering OS-9". It contains valuable software. DO NOT neglect to back it up. Instructions are provided in the first tutorial to help you.

To conclude, we hope you find that learning OS-9 Level 2 has never been easier. Enjoy!

DAY ONE: The OS-9 Ethos

Beginning Day One: The spirit of OS-9

The ancient Greeks imagined that every object in nature contained an ethos, a characteristic spirit which guidcd the ohject in the course of its existence. The lithe wall of a horse, the integrity of a dramatic play, the sonority of a lyre, all were exmples of ethos. The word embodies excellence and elegance, balance and focus, the marriage of thought and actlon.

Today, we don't have a word to describe ethos. And yet it still exists, especially in software such as OS-9, a product designed with intelligence and implemented with care. What is the ethos which the Microware engineers provided OS-9? To find out, use this book. It starts you on your path to understanding the OS-9 ethos in its usefulness and elegance.

What we'll cover

You may never have to know the details of how OS-9 works. Still, we now take the time to show you the broad outline of what OS-9 does inside your CoCo. We introduce terms to help you understand configuring a custom system disk, terms such as pipe, driver, and descriptor.

In addition, we explain in further detail what a command is, what it does, and where on a disk to store one. You learn ahout data and how to organize it in logical directories. We also introduce general concepts which clarify the OS-9 ethos.

The OS-9 Boot Process

Starting software on a computer, by long-standing tradition, is called booting that software, and OS-9 is no different. Part of the code which boots OS-9 is contained on your system disk in a file named OS-9 Boot.

The OS-9 boot process installs information in your computer which it needs to deal both with "higher-level" humans and "lower-level" computer hardware. The low-level functions of OS-9 involve managing your computer's memory and peripherals (peripherals are devices attached to your CoCo such as printers, modems, and disk drives). High-level functions of OS-9 accept your English-language commands, interpret them, and see to it they are executed.

The Kernel and I/O Manager

The Kernel and the I/O manager (IOMan) handle most of the low-level activities of your OS-9 computer. The Shell is in charge of most high-level activities. Following is a brief summary of each of these important parts of OS-9:

* The Kernel, the heart of OS-9, manages computer resources such as memory, requests for input and output of data, and coordination of all programs, allowing multi-tasking.
* IOMan is the "errand boy" for the Kernel during data input and output; it can recognize and "talk to" all different kinds of peripheral devices.
* The Shell accepts your English language commands and translates them into a form the Kernel can understand; the Shell is the courier between you and the Kernel.

Of these three parts of the OS-9 operating system software, the one you encounter most often is the Shell. It prompts you to give it a command by printing "OS9:" to your screen, a prompt you wlll find familiar— and friendly— by the end of this book.

OS9Boot

When you power up your Color Computer 3 in preparation for OS-9, there is already software purring inside: Disk Extended Color Basic. Your first step is to replace this software with OS-9. Disk Extended Basic uses the DOS command to find the OS-9 Kernel which in turn finds OS9Boot. Both are loaded from the disk's storage area into the CoCo 3's internal storage area, called memory. OS9Boot and the Kernel are found on an OS-9 System Disk (sometimes called Boot Disk after the OS9Boot file it contains).

OS9Boot is one example of an OS-9 file. A file contains a series of patterns (usually electric or magnetic) which OS-9 understands. The precise structure of the OS9Boot file can vary. It is composed of modules of codes which can be mixed and matched in many ways. These modules contain information OS-9 uses to run your computer and its peripheralls— disk drives, ram disks, printers, and so on. Because OS9Boot is a file many users casually call it a bootfile.

When you add a new peripheral, OS-9 can talk to it only if you also install new modules into memory. Even though you may easily load these modules at any time, most people include them in the OS9Boot file. Since OS9Boot is present in your computer from the moment you Mastering OS-9, the modules for your new peripheral will be ready for you whenever you need them.

Creating your own, personal OS9Boot.

Creating OS9Boot files to match your system is not a trivial task. It is also not an impossibly complex task— when you know what you're doing. "Mastering OS-9" tells you how. In fact, we show you three ways to make a bootable system disk. When you master the craft of making OS9Boot files, you master your

computer. Understanding each method of OS9Boot creation involves the same terms; modules, and bootlists, for example. Keep an eye out for these two.

Each way to create an OS9Boot shares two other traits. First, the 0S-9 Kernel is written on part of a system disk's Track 34. A track on a disk is somewhat like an individual song on a record album. Specifically, it is a concentric ring on your disk which is set up by OS-9 to store data. Just as you must place a phono needle in just the right place on a record album to hear the Beatles' "Eight Days a Week", your computer must also place the disk drive head at the correct place on the disk to find the Kernel. This is the general function of the Disk Extended Color Basic DOS command.

The second feature common to system disks is that OS9Boot is always created as an unbroken file on the same disk as the Kernel. Because the boot process is picky about the existence and locations of OS9Boot and the Kernel, simply copying over the OS9Boot file from one disk to another is insufficient to make a disk bootable. You must learn to correctly craft your own OS9Boot files.You should be comfortable with each of the three ways of creating bootfiles by the end of this book. Each one allows you to customize your system to differing degrees.

Memory modules

The idea of adding modules to a bootfile excites knowledgeable computer fans. Imagine: an operating system that's expandable and easy to customize! Plus, correctly-written modules can be added to memory even after bootup. This practice is handy when testing a new peripheral or adding a ramdisk drive on the fly, for example. OS-9's flexlbility and fame stem from this revolutionary memory module system. It stands as one of Microware's proudest achievements.

There are a few practical limits to customizing OS-9's modular boot related to particular hardware and software combinations. Most users never reach these limits, but if you do, we outline causes and solutions to these limits during our discussion of Config in "Getting Started With Tutorial 1".

Bootfiles are easy to understand and create

The foundation for OS-9 is the OS9Boot file. Once the bootfile's role and structure are clear to you, other OS-9-related concepts will fall into place.

OS-9Boot Summary

SUMMARY OF OS9BOOT SECTION

* The OS-9 Kernel and the OS9Boot file (sometimes just called bootfile) is the first OS-9 software loaded into your computer.
* The bootfile contains modules which allow OS-9 to function and to talk to your peripherals.
* You can affect which modules a bootfile includes by using three different techniques.
* These techniques always create the bootfile and Kernel as the first step in creating a bootable system disk.
* You must learn at least one of these techniques in order to customize your OS-9 computer.

The **kernel --**
OS-9 is ticking away

The OS-9 Kernel

The Kernel, a small and powerful part of OS-9, serves as a heart for your computer. The Kernel organizes your computer's resources to fulfill requests by you and your software. It know the "nuts and bolts" about which parts of your system are ready to do work. It knows how much memory is avallable, how much memory your programs need, and how much priority each program receives. It knows what devices are attached to your computer and the general kind of data each device requires.

Taking turns means multi-tasking.

One of the most useful and unique aspects of OS-9 is its built-in multi-tasking. Multi-tasking is possible thanks to the hardware design of your Color Computer and to the Kernel's software.

A computer can execute hundreds of thousands of instructions per second. Not all of these instructions do "useful" work. For example, most programs on most computers spend much of their time waiting for the user— you— to enter information at the keyboard. The programs stop a CPU from doing useful work while waiting for you input.

Under OS-9, however, as you enter information for one program your CPU can take turns serving other programs in memory. By switching between programs as quickly as ten times per second, the Kernel gives the impression that all your programs are running at the same time — and it hardly ever loses a keystroke of yours! (Actually, if the Kernel ever misses a keystroke it is not the fault of OS-9 but the hardware on which it runs. You can alter your Color Computer 3 setup to virtually eliminate lost characters.) OS-9 allocates your CPU's resources with efficiency not to be found in Disk Extended Color Basic.

Process Priority

Typically, when you use several programs (or processes) at once on your CoCo, each receives equal attention by the CPU; they have the same priority. If desired, you can boost the relative priority of any process at any tlme.

Why would you need control over process priority? Consider a typical scenario. Suppose in one window (a display screen created by the CoCo 3 which can present a program to you) you are word-processing. In another window, your spell-checker sifts through a large text document for spelling errors. The word-processor in the first window interacts directly with you making it a high-priority program. After all, you want your keyboard to react quickly to keypresses. In the other window, the spell-checker's text-sifting hogs the CPU and thus may take away some of the responsiveness of your word-processor. Typically, you don't care if the spell-checker finishes in thirty seconds or a minute. This makes the spell-checker a low-priority program. Simply increase the priority of your word-processor and decrease the priority of your spell-checker. The spell-checker works blissfully in the background while your word-processor doesn't even slow down.

*Commands you will use— **procs** and **setpr***

To increase the priority of a process, you first must know its Procss ID number. OS-9 gives each process a unique one. At the OS-9 prompt, type:

OS9: procs <ENTER>

and OS-9 will give you a report like:

```
User                 Mem Stack
Id  PId Number Pty Age Sts Signl Siz  Ptr  Primary Module
--- --- ------ --- --- --- ----- --- ----- ---------------
  2   1      0  128 129 $81  228    3 $02FO DEAD
  3   4      0  128 129 $80    0      3 $71E2 Shell
  4   0      0  128 128 $80    0     33 $75E2 gshell
  5   4      0  128 128 $80    0  119 $69B2 ds
  6   3      0  128 128 $80    0      6 $05F3 Procs
```

The left-most column shows the Process ID number. To increase the priority of the word-processor (named "ds" in the right-most column) to a value of 200, one would type:

OS9: setpr 5 200

The highest priority available is 255.

A slice of time.

The Kernel keeps track of how much time the CPU spends on each process by counting the numher of ticks that go by on its clock. Each tick is 1/60th of a second and is sometimes called a time-slice.

The Kernel's clock serves other purposes as well. Each time you boot up OS-9, the Shell asks you for the current date and time. The Kernel remembers this time and, every time you save a file to disk, the current date and time are saved to disk with the file. You don't normally need to know a file's last revision date, but it might come in handy if you ever want to find the latest version of a business proposal or form letter.

Kernel Summary

SUMMARY OF KERNEL SECTION

* The Kernel manages the system's resources.
* The Kernel divides the CPUs attention between different programs or processes.
* You can change process priority.
* You can — and should — set the time for your system.

Unified I/O

IOMan: Input and Output Unification

The Kernel, busy with multi-tasking and with servicing your programs, leaves a lot of the "dirty work" to the input/output Manager (IOMan). IOMan handles the finer details of transferring data around your computer system. This design displays one of the elegant features of OS-9: unified input/output. This concept has been notably included in the UNIX operating system from AT&T. OS-9's implementation of Unified I/0 continues to draw praise from many industrial giants.

Briefly, Unified I/0 makes files transferrable to or from any device or program on your computer. The precise nature of the data itself is irrelevant to the operating system (in our case, OS-9). It is up to the destination device or program to be able to make sense of the data. Unified I/0 thus helps OS-9 be device indpendent. If the data to be transferred contains control codes (say, for on-screen formatting), it is up to the program to interpret the codes and provide the actual formatting. The consequence of Unified I/0 which concerns us is that, under OS-9, you can often direct data output to and input from many different devices and programs. This increases the flexibility and power of the software and hardware on your system .

IOMan's assistants — RBFMan, SCFMan, and PipeMan.

The precise way IOMan sends data around your system depends on the device to which the data is being sent. Some devices can only handle one character at a time. Others can handle whole blocks of characters at a time. Additionally, a device can be a virtuall device — not a physical device but part of your monitor screen or memory which can be configured to act like a real, physical device.

To help IOMan transfer data to a device there must exist two modules in memory: the device descriptor and its device driver.

More on device descriptors

IOMan uses the device's descriptor to find out which way to send the data.

* If the device can handle blocks, IOMan passes the data to the part of OS-9 which manages blocks (the RBF Manager). An example of this kind of device is your disk drives.
* If the device accepts one character at a time, the SCF Manager moves the data to the device (for example, to a modem or a printer).
* Another data manager deals exclusively with the virtual device called the pipe. Pipes offer a way for different programs to pass data to each other. IOMan calls on the pipe manager PipeMan for this task.

When you learn to configure a custom OS9Boot, you will see the terms RBF, Pipe, SCF, and other related terms. Now that you know what they are, the configuration process will be less of a mystery.

Device descriptors also include the name of the device as OS-9 will use it. Your printer is typically named "/p1". The slash, "/", tips off OS-9 that "p1" is a device and not a command or other module. Your first disk drive is typically named "/d0".

Device descriptors can also contain numbers — initializing values — which help OS-9 "talk to" the device correctly. Windows, for example, can be created in many different shapes and sizes. The window descriptors arrive on your system disk with initial size and shape information. We will continue to explore disk drives, modems, printers, pipes, and windows as "Mastering OS-9" progresses. If these concepts are still hazy to you, no need to worry.

Device Drivers

OS-9 requires each device to have a device driver and descriptor. As noted above, the device descriptor describes to OS-9 many general features of the device — its name and the type of data it requires, for example.

The device driver is much more specific. It provides OS-9 with the ability to communicate with a device in the device's own language. The device driver, in some ways, is OS-9's doorway to the real world. It knows the nitty-gritty, ugly details of how to communicate with a peripheral. Examples are the drivers for your disk drives, your keyboard and screen, and your printer.

How you will use descriptors and drivers

Device descriptors, in conjunction with their drivers, allow OS-9 to send its "unified" I/O to all the peripherals on your system. Many device drivers and descriptors also allow the peripherals to talk to your computer.

When you attach a new device to your Color Computer 3 you must also install the device's driver and descrlptor modules. This installation can be accomplished in several ways. The best way, mentioned in the introduction to OS9Boot, is to include these modules in your OS9Boot file.

IOMan Summary

IOMAN SUMMARY

* OS-9 uses the Unified I/O philosophy made popular by UNIX.
* The unified I/O of data is managed by IOMan.
* IOMan determines whether each devlce attached to your computer accepts data in blocks, in sequence, or in a pipe.
* IOMan then delegates data transfer to RBFMan, SCFMan, or PipeMan as necessary. These modules must be in your OS9Boot file.
* IOMan uses device drivers to communicate with your peripherals.
* Every device attached to your system requires a device driver and descriptor.

The OS-9 Shell

The Shell: Guide and Intermediary

Except for terms associated with the Kernel and lOMan — terms such as priority and driver — you need to know very little about these two parts of OS-9. The reason is that OS-9 provides an intelligent lntermediary — the OS-9 Shell. The Shell, a powerful OS-9 program, has several useful functions:

* The Shell translates your keyboard instructions into special, compact codes which the Kernel requires to do its work. If your keyboard instructions are mistyped or misguided, OS-9 offers you a numbered error message to help you find your mistake.
* The Shell can also accept instructions written in text files. We'll examine procedurefiles in detail during the tutorials.
* The Shell's command-line modifiers make it easy for you to start several tasks. You can run one command after another (sequential execution), or execute them simultaneously (concurrent execution). You can also execute a program and direct its output to another program or to a data file (input/output redirection). See the following section on "Paths".

After you boot OS-9 and enter the date and time, the Shell comes to life on your terminal screen. When it's ready for your commands, the Shell prints "OS9:" on your screen, like this:

OS9:

This symbol, called a prompt, is the Shell's way of asking for an instruction. To the right of the prompt is a graphics block called the cursor. It marks the current position where text appears as you type on the keyboard.

Once you enter your instruction at the prompt, press the key marked <ENTER> . The Shell now accepts your entry and attempts to interpret it. If you misspelled the entry, OS-9 discovers the mistake and prints an error message to your terminal screen.

What to enter: Words that do work.

OS-9 generally understands two types of entries at the "OS9:" prompt. First, the entry can be the name of an executable file. Secondly, the entry can be the name of a procedure file. Let's quickly define these terms before moving into a more in-depth discussion of each.

Executable Files

Executable Files — Executable files contain computer code which your CoCo 3 can interpret directly (although it may sometimes need the help of a run-time module as with BasicO9). Most OS-9 users don't employ the term "executable file" referring to one instead by one of the following terms: command, utility, program, or application. You cannot read or alter the code in executable files without powerful software tools and some programming knowledge.

Executable files can be kept in your CoCo's memory and on disk. If an executable file is in memory it is executed immediately. They are often followed by names of data files, as when a word processor (an executable file) is followed on a command line by the name of the file you wish to edit.

Procedure Files

Procedure Files — You can read a procedure file by listing it to your screen or printer. It is a text file, created with any text editor, which contains the names of valid OS-9 commands. It may also contain the names of other procedure files. In addition to reading procedure files, you can also edit them or create your own from scratch. Procedure files are stored on disk. When you enter the name of one at the "OS9:" prompt it is executed one line at a time just as if you were typing each line by hand. You use procedure files to automate repetitive tasks.

Executable Files: Commands, Programs, Applications, & Utils

OS-9 users have many different names for executable files. They differ only in code size or number of features and generally may be classified as follows:

A command typed at the "OS9:" prompt is the name of a small file of computer code. The Shell sees to it that the Kernel executes the specified file. The Kernel uses these command files to do most of your work.

A command and a utility are basically the same, although a utility is often used for a specific purpose such as counting the number of words in a text file or converting numbers back and forth from hexadecimal to decimal. Commands are often used for general purposes such as listing a file to your screen .

Commercial programs, or applications, can be very large indeed. These two terms are used interchangeably. The term applications seems to be more in vogue perhaps because the word conveys that computers can be applied to solving your problems. Calling a large executable file a program has perhaps been avoided because people in general dislike or are afraid of programmlng.

Applications usually provide a wide array of features. You often access these new features through techniques sometimes completely different than those employed using Shell. For example, most applications currently incorporate a mixture of menus, where you choose from a text list of options, and graphic icons, where you choose a program function by selecting an on-screen option with a mouse. Frequently an application avoids using Shell for many functions, instead using the Kernel directly. But a Shell incorporates so many useful functions that an application usually provides you a gateway to one.

Examples of applications are:
* word processors
* spell-checkers and on-line thesauruses
* telecommunications software
* databases
* spreadsheets
* stock portfolio managers
* graphics programs for drawing and desk-top publishing and other, more specialized products.

For a complete overview of these types of applications see the

essays at the end of this week's tutorials.

Procedure files: delegating a tedious j ob to the Shell.

Procedure files, or simply procedures, are a variation on executable files. They themselves are not executable code. They contain names of commands or other procedure files, ordered to accomplish a complex task. In a sense, a procedure file is a "script" such as an actor might use to learn his or her movements around a stage. The Shell uses these scripts as instructions for its activities around your system. UNIX users will recognize at once that procedure files parallel the UNIX Shell script. This term is even used in OS-9 circles in place of the term procedure file. If one is familiar with MS-DOS, a procedure file is similar to a "batch" file.

You may use a word processor to create a text file listing valid OS-9 command lines, one per line, and save it to disk. Just invoking the name of the text file at the "OS9:" prompt is sufficient to start the sequential execution of each included command line. Examples of procedure files will be discussed and dissected in later reading.

Paths and pathnames

OS-9's Kernel expects to send all output data to your screen. In OS-9 parlance, your screen is the standard output path. Error messages are also sent to your screen, making it serve double duty as the standard error path. Similarly, OS-9 normally accepts input from your keyboard making it the standard input path.

Just about every program you run under OS-9 uses these three paths. Why not? They're built into the Kernel ready for use by any program requiring keyboard input and computer monitor output. But if these paths are built into the Kernel, why mention them here in a discussion of the Shell? Because the Shell can *redirect* each of those paths.

Example: If Utility1 normally outputs data to your screen, the Shell can redirect that output to a disk file by using the Shell's reserved symbol for output redirection (>) like so:

OS9: utility1 > diskfile <ENTER>

If Utility2 normally accepts data input from your keyboard, the Shell can instead send it data input from a disk file when you use the reserved input redirection symbol (<):

OS9: utility2 < diskfile <ENTER>

You can often send the output of Utility1 as the input to Utility2. This uses the Shell's pipe modifier (!):

OS9: utility1 ! utility 2 <ENTER>

Tutorial 10 demonstrates how to redirect all three paths in order to automatically start your favorite program in a custom window.

Shell Summary

The Shell's presence between you and the Kernel makes running OS-9 fun and exciting. We'll get to know many of its useful options in the tutorials.

SHELL SUMMARY

* The Shell, an OS-9 program, exists as an intermediary between you and the Kernel.
* The Shell executes commands, procedure files, and applications when you invoke them at the "OS9:" prompt.
* The Shell has several options, allowing you to start processes either sequentially, concurrently, or utilizing input/output redirection.

Kernel, IOMan, & Shell Notes

NOTES ON THE KERNEL, IOMAN, & THE SHELL

1. Many applications handle both low-level and high-level computing jobs. For example, a typical desk-top publishing program may be written to handle the low-level tasks involved printing graphics even while it handles the high-level task of creating a news article.

2. There are many ways to speed up system performance. Setting process priorities is one. For example, I can word-process in one window while my spell-checker, in another window, checks another document. To keep my word processor feeling snappy, I boost its priority to 200 out of a possible 255. I lower the spell-checker's priority to 40. With these priorities I never feel a slowdown. The spell-checker checked a 2000 word document in one minute forty-five seconds — only 15 seconds longer than if had left its priority equal to the word processor's! No-halt floppy controllers, ramdisks, and hard disks are also good choices to speed up your CoCo 3's performance.

3. UNIX and similar operating systems try to make data transfer easy and it's paying off. Personal Computing recently surveyed industry leaders and found that UNIX's popularity was sharply on the rise.

4. Sony and Philips searched carefully for an operating system for their state-of-the-art Compact Disk-interactive (CD-i) devices. These devices are really super-microcomputers which have special support chips to output a multitude of graphics,

text, and audio formats. Their search ended with OS-9 due in large part to OS-9's device independence and easily-maintainable code. A proposal from Microsoft based on Windows was turned down. Microsoft made an offer to buy Microware shortly before developing their own alternative. Microsoft executives recognized Microware's lead in real-time operating systems.

Some program data needs formatting and other special codes to make sure data have built-in relationships. Data such as this, while easily transferred to devices and programs, may be to specific to a single application to be interpreted by the target programs and devices. Spreadsheets are a common culprit. OS-9 spreadsheets usually offer a way around this. Dynacalc, for example, offers a data file option which any OS-9 word processor could read.

It should be stressed that data files unreadable by devices or programs may not be violating the Unified I/O principle. As long as the operating system perceives the file as a valid system "object", Unified I/O is maintained.

6. Actually, OS-9 comes with serial printer support built right in. No need to buy any descriptors and drivers for your printer — as long as you plug it into your serial port directly or through a serial to parallel convertor. You can buy true parallel ports for parallel printers (no serial-to-parallel convertor required). Check "the world of 68' micros" magazine or Delphi and other on-line services (don't forget Internet and FIDO networks) for vendors who can help you.

Beginning Day Two: OS-9 Directory System

DAY TWO: ***Day Two:***

Introducing the OS-9 Directory System

When you finished preparing your taxes this year, you probably made a vow tobe more organized tax time next year . You can avoid a lot of pain if you keep an on-going filing system.

You might first create a simple filing system making it easy to file each receipt as you go. You would also make it flexible to allow for changes.

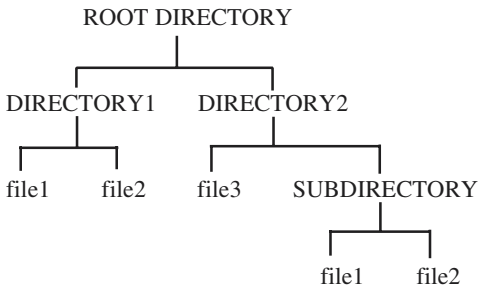
Using file cabinets, drawers, file folders, and envelopes provides enough ways to sort your receipts and records. Use a whole file cabinet for all of your deductibles, and then use the drawers in the cabinet to hold specific kinds of deductibles such as travel expenses, mortgage payments, and so on.

Then, within the drawer holding travel expense records, you may have separate file folders for different trips (or maybe not, choose any organizing principle that suits you).

Files and folders parallel files and directories.

OS-9 uses a directory system as intuitive as this. Called a hierarchial directory system, it has parallels to drawers and file folders. File folders can even be within file folders! OS-9 calls these logical partitions directories, subdirectories, and files.

Files are the smallest unit. Files are put in subdirectories, and subdirectories are put in directories. A directory can hold names of subdirectories and files; one or several of the subdirectories may contain the names of files and other sudirectories (which are, thankfully, not called sub-sub-directories). This system is both simple and flexible. Most OS-9 users cannot imagine living without hierarchical directories; most newcomers to OS-9 cannot imagine them at all. Study the diagram below. It outlines only one possible directory structure that can be created.



Root Directories

In this diagram, the root directory is the topmost logical component in the structure. It "contains" every other file and directory on the disk. Even though the diagram shows the root directory's name to be "root", a storage medium's root directory always has the same name as the storage device in which it currently resides. For example, if OS-9's name for your boot-up drive is "/d0", then any floppy inside that drive has a root directory named "/d0" as well. Similarly, disks in drive /dl have a root directory named "/d1".

Directories and Subdirectories

The only two names contained in the root directory shown are DIRECTORY1 and DIRECTORY2. These are the components of the directory structure just one level down from the root directory. The contents of DlRECTOY1's directory are just file1 and file2, but notice that DlRECTORY2 contains yet another directory, called a SUBDIRECTORY (a directory within a directory). Note that directories directly under a root directory are not usually refferred to as subdirectories.

Directory levels below a root directory cause the directory structure to bloom, and the branches and leaves are directories and files. It's no wonder that a directory structure is often called a directory tree! In the first few tutorials you gain experience moving around a directory structure and creating one of your own.

Files

The basic unit of your work is the file. OS-9 users most often deal with files which fall into two fundamental categories: executable files such as commands and applications, and data files such as letters to prospective clients and lists of phone numbers. Procedure files are also data files even though they can he "executed" in a sense. Both executable files and data files must have unique pathnames. We discuss pathnames in the next chapter.

Executable Files

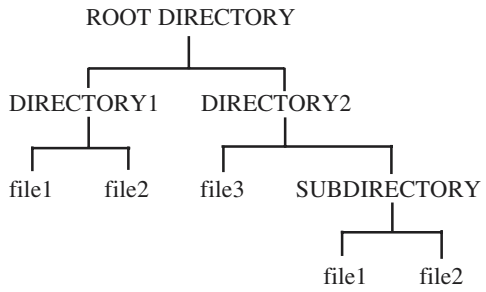
Executable files contain special codes designed for direct reading by the Kernel and your CPU. You cannot read these files. Data files usually contain characters which are acted on by executable files. For example, word processors (executable files) create letters and proposals (data files). Likewise, graphics processors create pictures (graphics files).

In a large sense, executable files are like verbs — they take action of some kind. They frequently take action on data files, which are thus like nouns. OS-9's directory system philosophy separates the "nouns" from the "verbs". To be precise, executable files used for commands and applications are kept in an

execution directory. Data is kept in data directories. We discuss these in the pages ahead. As you might guess, organizing your data and executable files in directories takes planning. The planning is well worth it.

Before discussing directories and how to plan them in further detail, we introduce path names. Path names provide shorthand directions used by OS-9 to navigate through the directory structures you create.

The OS-9 directory system is hierarchical; that is, a directory may contain other directories. A directory which contains files or other directories is called a parent directory. Parent directories may give rise to a family tree of directories. Each directory in the directory tree may contain files or other directories, and so on to any level of organization. Examine our directory structure example from the previous section:



The root directory is the parent directory to DIRECTORY1 and DIRECTORY2. DIRECTORY1 is the parent directory to File1 and File2. DIRECTORY is the parent directory to File3 and SUBDIRECTORY. Each parent directory contains all files and directories below it in the directory structure. Thus, DIRECTORY2 contains File3, SUBDIRECTORY, and both of SUBDIRECTORY'S files. To continue the family tree metaphor, you can accurately distinguish between File1 within DIRECTORY1 and File1 within SUBDIRECTORY if you specify its parent and — if you will —grandparent. A full name for each of these File1s must include its full "lineage".

To phrase this in OS-9 parlance, every file on a disk has a full path name. This path name explicitly states the path OS-9 takes to find the file. A full path name contains three sections:

1) the name of the root directory,
2) the name of subdirectories, under the root directory, which contain the file, and
3) the local name of the file.

The general syntax for a full path name is:

/[rootdirectory] /[subdirectory] /... /[subdirectory] /[local.filename]

Local file names are defined shortly. In the meantime, we examine the first two parts of a full path name. After the above syntax is clearer, we will show some examples.

Root Directories

Each mass-storage device (floppy disk drives, hard disk drives, and ramdisks) has a root directory with the same name as the device name. When you specify a device on the command line, OS-9 looks for the device name in a list of device names which it keeps in memory. For most purposes, you type the device's name with a preceding slash.

For example, imagine a data disk in Drive /d0. The name of the device is /d0, where the slash signals to OS-9 that /d0 is a device. Similarly, /p is "the device p", /term is "the device term", and so on.

The name of the root directory of any disk in Drive /d0 is /d0 as well. To list the contents of a device's root directory, simply specify the name of the device after the Dir command. Since all full pathnames begin with a root directory name, we can formulate the following rule from what we have learned: When specifying a full path name, precede the root directory name with a slash.

Current data directories

Just as you might use a finger to keep track of your place in a book, OS-9 uses current data directories to keep track of your "place" in the directory structure. In the book analogy, you might say to yourself "I want to find a photograph I saw in the chapter I'm currently reading". Calling that chapter a "current chapter" is much simpler, for example, than calling it "the chapter which follows the fourth chapter which follows the third chapter which follows the second chapter...". In OS-9 parlance, you establish a current data directory to avoid constantly specifying the complete path name of the directory or file in which you
are interested.

Commands you will use — **chd**

When you first boot OS-9, your default current directory is usually the root directory of drive /d0 (a hard drive user has /h0 or some similar name as the default current directory name). You can change which directory is your current directory with the **chd** command.

If every person's name included his or her entire lineage, you'd go crazy. Imagine the size of the "Name" blank on application forms! Imagine going to a party and seeing a "Hello My Name Is" sticker that reached out the door and down the elevator!

With OS-9 you can avoid continually typing the full path name for a file. Just change your current data directory to the directory which contains the file and then just type the local file name.

Local files and subdirectories

You can get a list of file names in the current directory by typing:

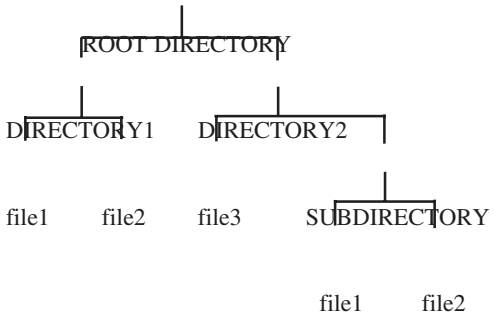
OS9: dir <ENTER>

The resulting text on your screen only gives the names of the files located in the current directory. The FULL path name for a file on your screen is merely the path name of the current directory with the file name attached to the end. The Dir command does NOT return complete path names for each file — and thank goodness, since those path names can be long indeed!

For the puroses of "Mastering OS-9" we created a term for file names which appear ln a current directory listing. The term is local file name. A local file name is the name of a f ile as it appearsin the current directory. Thus, if your current directory is /d0, the local file name of /d0 /OS9Boot is simply OS9Boot. Similarly, a local directory name is the name of a subdirectory appearing in the current directory listing. Thus, if your current directory is /d0, then the local directory name of /d0 /SYS is simply SYS.

Now that these terms are defined, here is an important rule specifying items in the current directory: Local file names do not need a slash preceding the name. Similarly, local directory names do not require a slash preceding the name.

Let's now show the chd command in action through imaginary examples. Look again at our sample directory structure:



Example: Listing of the root directory contents (see chart) would show only DIRECTORY1 and DIRECTORY2. If you set your current data directory to the root directory, typing:

OS9: dir <ENTER>

displays:

Directory of . 16:29:56
DIRECTORY1 DIRECTORY2

to your screen.

Once you use the Chd command to change your current directory
to DIRECTORY1, a listing of the directory contents shows:

OS9: dir <ENTER>
Directory of . 16:31:
file1 file2

Example: File1 under DIRECTORY1 can he called simply File1 if you first change your current data directory from the root directory to DIRECTORY1. If your root directory is /d1, first type:

OS9: chd /d1 /directory1 <ENTER>

Now you can type:

OS9: list file1 <ENTER>

and OS-9 knows to list /d1 /directory1 /File1 and not, say, /d1 /directory2 /subdirectory1 /File1. You may also type:

OS9: list file2 <ENTER>

to list /d1 /directory1 /file2.

Example: After a long session at your computer, you may have used the chd command several times to move yourself about the directory tree. In order to see the root directory of drive /d0 without reassigning the current directory, type:

OS9: dir /d0

This displays a root directory listing. The name /d0 is the full

path name of the root directory of the disk in drive /d0. No matter where you position yourself in a directory tree with chd you may always find a directory listing of any directory by typing dir followed by a full pathname of the directory whose listing you desire.

Example: Assuming you, like many CoCo OS-9 Level 2 users, don't use a hard disk yet, if you type:

OS9:dir <ENTER>

right after booting up, you will see the root directory of drive /d0 displayed on your monitor. These file names and directory names are local to the root directory. You can receive the same listing if you type:

OS9:dir /d0

You may have surmised from previous reading the following rule: Slashes are used to separate directory names from each other in long path names. The only two occasions when you must type a preceding slash occur when you specify a device name (such as /p, /term, or /d0) or when you must specify a full path name starting with the root directory.

Example: In my /d0/SYS directory, I have a file whose local file name is df.init. Its entire path name is /dO/SYS/df.init. Imagine my current data directory is the root directory of drive /d0. To list the file df.init, I have several choices:

1) I can change my current directory to /d0/SYS; in this case I type: **OS9: chd /d0/sys <ENTER>**
OS9: list df.init <ENTER>

2) I can use the local directory name rule and type:
OS9: list sys/df.init <ENTER>
(Notice there is no preceding slash)

3) Or I can specify the entire path name.
OS9: list /d0/sys/df.init <ENTER>
(Here you see a preceding sIash since /d0 is a root directory name)

Anonymous directories

Every file on an OS-9 disk is contained within one and only one parent directory. Each directory except the root directory is also contained within only one parent directory. This is similar to saying that each of us has only one set of biological parents.

OS-9 uses this fact to help you find a directory listing of a parent directory, a "grandparent" directory, and so on, without having to specify each path name. If a dot (.) represents the current directory "generation" (your parent directory), then dot-dot (..)

represents the previous "generation" (your grandparent directory, if you will).

To demonstrate, typing:

OS9: dir . <ENTER>

dislays the contents of the parent directory (typing just dir at the"OS9:" prompt gives you the same result so most users never bother typing this command line). Similarly,

OS9: dir .. <ENTER>

shows you the contents of the grandparent directory. You will see the name of your current directory in that listlng. You can also use the "dot" and "dot-dot" method to change current directories.

If you are at the very bottom of an immense directory tree, you can move to somewhere in the middle and look around for your bearings with:

OS9: dir .... <ENTER>

Use several dots. lf you see the name of a desired directory or file then type:

OS9: chd .... <ENTER>

using the same number of dots.

Naming previous generations of directories with dots and not with their full path names is easy and fun. Any directory named with dots and not its "real" name is called an anonymous directory. It might be advised that you avoid the anonymous directory convention until you get the hang of building and moving about complex directory structures. As you grow accustomed to directory structures you will find anonymous directories convenient.

Pathname Summary

PATH NAME SUMMARY

The rules for putting all the names together:

* The general syntax for a full path name is
  /[root directoryl /[subdirectory] /.../ [subdirectory] /[filename]
* Full path names for a file potentially include the names of many directories. They always end with a file's local file name.
* Slashes separate directory names from each other and from the local file name.
* lf the path name begins with the name of a root directory then you are giving the full path name. When giving a full path name
  precede it with a slash. Otherwise, a path name should not begin with a slash.

Data Directories:

* Reading from the right, each file or directory is contained within the directory to its left. This is the basis for anonymous directories.

Data Directories: Keeping Data Organized

In this section we explore how to use OS-9's hierarchical directory system for data storage. This directory system provides a method for organizing data in a way that is sensible and easy to use. You learn about the Makdir and Deldir commands and exercise your knowledge of pathnames.

When you use an application you usually create data files. Using a single application (a word processor, for example) you might produce hundreds of separate data files. With several different applications the total number of data files may eventually reach into the thousands. How wil! you go about organizing all these data files so you can get to them qulckly?

Commands you will use— **makdir**

OS-9 offers data directories and subdirectories for exactly this purpose. Just as you might store related personal papers in file folders and related folders in drawers, OS-9's directory structure relies on the notion that logically related files should be placed together. Because you can create directories and subdirectories in any manner your work style requires, storing your data files is easy and flexible.

Suppose you have just finished booting OS-9. Usually OS-9 establishes the root directory of drive /d0 as your current data directory. Also, /d0/CMDS is established as the current execution directory.

Suppose you just purchased Ulti-U-Til, "the Ultimate Utilities," from XYZ Corporation. These utilities are executable files similar to the OS-9 commands which arrived on your System Mater from Tandy. Suppose also that you wish to put these files in your CMDS directory but you want to group them together.

Create a subdirectory of /d0/CMDS called, say, /d0/CMDS/XYZ. To do this, use the makdir command. Begin by typing makdir followed by the path name of the directory which will contain your new directory (in this case, we would type "makdir /dO/cmds"). Then type a slash ("/") followed hy the name of your new directory in capital letters. Then press <ENTER>. Altogether, here is what you would type:

OS9: makdir /d0/cmds/XYZ <ENTER>

Distinguishing between directories and files

Now copy the utilites from the Ulti-U-Til disk into your new directory. If one of the utilities is named dsort, it now has the pathname /d0/cmds/xyz/dsort. To use this utility all you have to type is: **OS9: xyz/dsort**
OS-9 can now find the command (how OS-9 finds commands is discussed in more detail in the next chapter).

Organizing files in directories.

Notice that the directory XYZ was created with all capital letters in its name. By common agreement, OS-9 users use capital letters for directory names and lowercase letters for file names. A mixture of capital and lowercase letters is sometimes used for file names, but NEVER for directory names. Your OS-9 life will be much happier if you follow this simple rule. If you don't, it is impossible to tell the difference between files and directories in a normal directory listing. In the tutorials we give you plenty of practice creating directories and files, so the "rule" should be a "habit" by the end of "Mastering OS-9".

Organizing your files into directories helps. Let's experiment in our minds to learn this organizing process. Imagine that the task at hand is to organize your word processing files.

The first step is to determine how you classify your work. For example, the results of your computer productivity might fall into two broad categories: letters resulting from business and letters relating to your personal life. Once you establish the logical categories of your work, prepare a disk with corresponding directories. For example, as you work with a word processor in drive /d0 you may store all your data on the disk in drive /d1 (the disk in /d1 will only contain data and thus becomes your data disk). You can create two directories on your data disk with makdir which parallel the categories business files and personal files.

Example: Once you format the data disk for OS-9 (see Tutorial 1) place it in drive /d1 and type:

OS9: makdir /d1/PERSONAL <ENTER>
OS9: makdir /d1/BUSINESS <ENTER>

Letters to friends and family you save in /d1/PERSONAL. Business related letters would be stored in /d1/BUSINESS.

If your business work requires both letters and business proposals, create subdirectories:

OS9: makdir/d1/business/LETTERS <ENTER>

Directory names.

OS9: makdir /d1/business/PROPOSALS <ENTER>

The full name of a directory specifies the path along the directory tree which OS-9 takes to find that directory. Likewise, a file within a directory has a full name corresponding to the full path OS-9 follows to find it.
In the example on the previous page, the data disk's root directory has the name /d1. Creating BUSINESS on that disk makes the full name of the BUSINESS directory "/d1/BUSlNESS". What happens if you moved the disk over to drive /d0?

Unique names required.

First, signal OS-9 that you have moved the data disk over to drive /d0 by typing:

OS9: chd /d0 <ENTER>

The BUSINESS directory now has the full name "/d0/BUSl-NESS". Subdirectories within the BUSINESS directory, LETTERS and PROPOSALS, now have new full names too: "/dO/BUSlNESS/LETTERS" and "/dO/BUSlNESS/PROPOSALS".

Data directory summary

Being able to change your current data directory means you often can avoid stating a file's complete path name. Still, OS-9 is always aware of each file's full name. To avoid confusing files, OS-9 requires unique path names for each file and directory. This requirement prevents you from creating a file or directory with a path name that is already used on a disk. If you try, OS-9 sends you the error message "Error 21" ("File Already Exists") to your screen.

DATA DIRECTORY SUMMARY

* Create a directory structure which reflects the way you organize your thoughts and papers.
* Use the makdir command to create a directory. If the directory you create is not in the current directory, provide a path name sufficient to have OS-9 place it within the correct directory.
* Use all capital letters when creating a directory.
* OS-9 prevents you from creating file or directories with duplicate names.

Execution directories

Execution Directories: Where the Action Is

During the boot process, OS-9 loads a large number of commonly-used commands into your CoCo 3's memory. When commands are in memory, the Shell can execute them instantaneously. Lesser-used commands are not loaded in memory. When you request a lesser-used command, OS-9 looks to a disk drive to find it. A floppy disk drive can hold hundreds of com-

mands. While access to floppy disk is slower than to memory, floppy disk drives are fast enough for most purposes.

To speed up OS-9's search for the command you want, Microware designed OS-9 to look for disk-based commands in a standard location — the current execution directory. Most floppy disks have at most one execution directory. Within that directory you may keep a large number of commands and other executable files. Bootable system disks have a directory named CMDS which OS-9 establishes as its current execution directory on
boot-up.

How and when Shell uses current execution directories.

If you guessed that the term "current execution directory" implies that a disk may contain additional execution directories, you were right! A disk may contain several directories each full of executable files. Execution directories need not be named CMDS and they may be placed anywhere within a directory structure. You will create execution directories most often in a floppy disk's root directory.

When you enter the name of a command at the "OS9:" prompt, the Shell looks first in memory. If the Shell cannot find the command in memory, it looks for the command name in the current execution directory. When it finds the command there, the Shell loads it into your CoCo and executes it. If the command you want is not in the current execution directory, it may be in another directory full of executable files. For OS-9 to find the command there automatically you must change your current execution directory.

When you change to a new execution directory, all commands you ask for — if not in memory — should be in that execution directory. If OS-9 fails to find the command it sends an error message (usually ERROR #216 — path not found) to your screen to let you know. A common reason first-time OS-9 users receive errors is that they replace the system disk with another disk after bootup. Since OS-9 expects to see the execution directory /d0/CMDS in a particular location, this situation confuses OS-9, even if both disks contain a directory named CMDS. You need to reorient OS-9 to the location of the new CMDS directory. To do this, use the chx command.

Commands you will use — **chx**

We now examine how you can change your execution directory. Suppose you have a directory named MYCMDS in the root directory of drive /d0. Thus its path name is /d0/MYCMDS. Imagine it contains word processing related commands and a word processor named "WordKing". Suppose also that "WordKing" is invoked with the command:

OS9: wk <ENTER>

OS-9 won't find wk unless its current execution directory is changed to /d0/MYCMDS. You can do this with the following command line:

OS9: chx /d0/MYCMDS <ENTER>

Now OS-9 will look in the directory /d0/MYCMDS instead of, perhaps, /d0/CMDS to find the commands you enter at the "OS9:" prompt.

Example: To keep OS-9 informed about disk switches, this is what I do:

1) I replace my boot disk (currently in drive /d0) with my word processing disk.
2) I type: **OS9: chx /d0/cmds**
which tells OS-9 to go find a new execution directory on drive
/d0. If I then want to use a third disk containing yet another CMDS directory, I repeat the above steps:
1) I replace my word processing disk with the new disk.
2) I type: **OS9: chx /d0/cmds**
and OS-9 looks again for a new execution directory.

Conclusion.

Incidentally, if a data directory I need ALSO exists on the new disk, I use the chd command to reorient OS-9 to the new data directory as well.

Execution Directory Summary

You switch current execution and data directories throughout the
tutorials. By the end of the book you'll be an old pro at keeping up with execution and data directories.You will also be on the way to using them to organize your data and programs logically, making you more productive and making your OS-9 computer a joy to use.

EXECUTION DIRECTORY SUMMARY

* A boot disk contains an execution directory named CMDS.
* OS-9 looks in execution directories for commands. If the execution directory is your current execution directory, the Shell loads commands automatically from that directory.
* There may be several execution directories on a disk but only one current execution directory.
* Execution directories may have any name and may be placed anywhere within a directory structure. Most often they are

within a floppy disk's root directory.

Command Syntax

You now know to change execution directories:

* Whenever you change from one disk which contains needed commands to another disk which contains needed commands
* Whenever one disk contains several execution directories and you need to switch OS-9's attention from one to another.

Command Syntax: Get It Right!

The fifty commands which arrive on the Level 2 System disk range from the simplest utilitv to the most complex. One command, date, displays today's date on your screen. In contrast, the edit command provides text editing, advanced macro capability, and multiple buffers allowing you to edit several documents at once with your own custom editing functions.

Despite this abundance and range of command power, you normally use only twenty or so commands. Two dozen of these commands, together with their options, can display great power and flexibility.

What we'll cover.

There is one catch though. Unless you type the commands and their options correctly, error messages — and possibly damage to your files — could result.

In the tutorials, we'll explain the pitfalls and dangers of some of the commands. Refer to the Tandy manual often, but when you do, use the information provided here to help you understand a command line's syntax.

We begin with a brief explanation of the following terms, complete with simple examples:

* command
* required parameter
* optionaI parameter
* options
* switches

Where OS-9 finds commands

Books about OS-9 adopt slightly differing notation when explaining command syntax. Keep this in mind when comparing this book's syntax with another books.

In the last section we touched on how the Shell looks for commands. Here we discuss that process in more detail. Once you

understand how the Shell searches for commands you will better grasp at what point the Shell, failing to find a command name, sends an error message to your screen. Being familiar with this process helps you deal intelligently with these error messages.

Recall that executable files are machine-readable codes which act in a useful way when correctly called from the Shell command line. Commands are one example of executable files. A command can exist in two places: in your computer's memory, and on a device like a floppy or hard disk drive. If OS-9 can't find a command name in memory it looks first in the current execution directory. If it fails to find the command name there, it looks in the current data directory. At this point, OS-9 has given up assuming that the name represented an executable file. Instead it looks in the current data directory for a data file such as a procedure file.

To sum up, when you type a name at the "OS9:" prompt, OS-9 takes up its search for the name in the following places:

1)Assuming the name is a command, OS-9 first looks in memory.

2) If it fails to find the command in memory, it next looks on your disk drive in the most logical place for commands— the current execution directory.

Commands in memory

3) If OS-9 fails to find the command in the current execution directory, it looks in the current data directory for a procedure file with the name you typed.

Don't put a command -- an executable file — in a data directory. Since you are in charge of where commands are stored on disk go ahead and put them in a directory with other executable files and chx to that directory when you need the commands it contains. Using similar logic, in most cases you will not put procedure files in an execution directory.

Commands you will use - **load** *and* **mdir**

We now examine the Shell's command search in more detail. We begin by discussing the advantages of having commands in memory.

The Shell can access in-memory commands immediately. Often-used commands can be placed in memory to improve your CoCo 3's responsiveness. Tandy and Microware had this goal in mind when they merged nineteen commonly-used commands into the Shell. After boot-up all these commandsreside in

memory along with the Shell, there for your immediate use (due to the CoCo3's hardware design, these extra nineteen commands take up no more memory than the Shell occupies by itself!).

If you need additional commands for a time-consuming, repetitive job, you may also use the OS-9 load command to load them into memory. We discuss the costs of using the load command in a tutorial later.

Modules loaded by OS-9 at boot-up.

OS-9 keeps track of which commands are in memory — and you can, too, by using the mdir command. Mdir lists the names of all modules in memory. This includes commands and modules such as device descriptors and data managers. An mdir listing of modules currently in my CoCo 3's memory is:

```
Module Directory at 14:33:20   <— Commands begin here
   REL      Boot     OS9p1
   OS9p2    IOMan    Init
   CC3Go    Clock    RBF
   CC3Disk  DO       D1
   DD       SCF      ACIAPAK
   T2       PRINTER  P        <— Some of these are not
   CC3IO    GrfInt   Term        commands but modules
   W        W1       W2          which install RAMdisks
   W3       W4       W5          on my system.
   W6       W7       PipeMan
   Piper    Pipe     GrfDrv
   Shell    Copy     Date
   DeIniz   Del      Dir
   Display  Echo     Iniz
   Link     List     Load
   MDir     Merge    free
   Procs    Rename   Setime
   TMode    Unlink   Ram
   RO       Rdisk    RamPak
   R1       gotoxy   ds
   grep
```

Speed is the advantage to having commands in memory. The drawback is that memory-resident commands take up more memory than you would expect, memory that is best used for big programs such as your word processor. The trade-off of speed for available memory can be handled easily with a few techniques. We introduce these techniques in Tutorial 6.

Example: BasicO9 (included with your Level 2 system) is a 24 kilobyte (24K) program which takes roughly 8 seconds to execute when loaded into my CoCo 3 from a floppy drive. If I load BasicO9 into memory first, it executes instantly. Loading BasicO9 from my hard drive takes a little under 5 seconds, and from a ramdisk the time is just under 4 seconds.

Commands on disk

Example: The cmp command which comes with OS-9 compares two files. If they are identical it reports no differences; if they differ, each character that differs is listed by number. This kind of file comparison is needed rarely so I usually don't load cmp into memory. Without cmp in memory, comparing two 6000-byte files takes five seconds. With cmp in memory, the comparison takes two seconds (the files I compared were stored on a ram disk and a hard disk).

Example: If my database software is in memory, I can start it in two seconds. If it is not in memory it executes in eight seconds off of a hard disk. By contrast, I can start it in six seconds off of a ram disk.

Unlocatable and misspelled commands

If a command does not appear in memory, the Shell must look in the current execution directory. If the command you requested appears there, OS-9 loads it into memory to execute it. When the job is finished, most often OS-9 removes the command from memory. This conserves system memory.

Commands you will use -- **error** *and* **help**

The Shell has a recourse if it can't find the name you type either in memory or in the current execution directory. The Shell tries the current data directory. If successful, the Shell reads this file into memory. However, the file is read as data and not as executable code. The reasoning behind this will be clear when we investigate procedure files in depth in the tutorials.

If the name you type at the "OS9:" prompt occurs neither in memory or in the current execution or data directories, OS-9 returns an error number to your screen (usually ERROR 216 — path not found). The precise error number provides a clue about what went wrong. Dealing with error messages can be frustrating, but Microware and Tandy provide two handy commands, error and help, to assist you along the bumpy error path.

The error command gives you a brief English-language explanation for the provided error number. For example, if I mistype list on a command line, like:

OS9: lust Daily Schedule <ENTER>

OS-9 will not be able to find a command named “lust” and would return a polite **ERROR #216**. At this point use the error command to get a fuller explanation:

OS9: error 216 <ENTER>
216 - Path Name Not Found

The help command provides a summary of a command's syntax. Just name the command after typing “help.”

OS9: help dsave <ENTER>

Required parameters

The screen will show:

```
Syntax: Dsave [-opts] [dev] [pathname]
Usage : Generates procedure file to copy all files in a directory system
Opts :  -b = make a system disk by using OS9boot if present
        -e <path> = make object disk using path as source
        -i = indent for directory levels
        -l = do dos process below the current level
        -n = do not include makdir commands in procedure file
        -s=num = set copy size to num K
```

Now that we have explored where the Shell finds commands, we tackle the specifics of command syntax. A command name, even if typed properly, may not function as you wish unless you also include additional parameters.

A parameter is a word or other symbol following a command on a command line. Some commands require a parameter. Most of the time these required parameters are file names or directory names, although sometimes parameters modify or enhance the function of the command. For the sake of this book, we distinguish between word parameters such as file names and options, switches, or modifiers. This latter category includes single letters, hyphens, or symbols used to increase memory allocated for a command.

Example: The list command requires a valid file name after it in order to function, and this file name is called the “parameter”. It makes no sense to type **OS9: list <ENTER>** because the Shell needs to know which file you wish to list.

Optional parameters

Example: Similarly, the copy command requires two parameters, the source pathname and the destination pathname. This, too, makes sense. When you call the copy command, the first name on the command line following “copy” is the file you wish to copy. The second name on the command line is the

pathname for the new copy of the file. For example:

OS9: copy ChapterOne /d1/LatestBook/FirstChapter

Options and Switches

Many commands permit the use of parameters in addition to any required ones.The additional parameters, while not necessary for the command to work, can save you steps.

Example: The del command has one required pathname, the name of the file you wish to delete. However, if you wish to delete several files, you may do so by listing the file name of each after the del command.

OS9: del file1 file2 file3 <ENTER>

For the del command, if all the files are in the current data directory use just the local file name. Any file not in the current data directory can be deleted if you provide a fuller path name.

In addition to parameters such as file names, you may type letters or numbers after the command; these are frequently called options or switches.

Example: The date command without options returns the current date: **OS9: date <ENTER>**

March 21, 1995

When you use the "t" option, you also get the current system time: **OS9: date t <ENTER>**

March 21, 1995 14:36:45

Memory modifiers

Example: Dir entered by itself at the "OS9:" prompt shows a listing of your current data directory to your screen. But type:

OS9: dir x <ENTER>

and you will see a listing of your current execution directory.

Example: The del command can delete commands from the current execution directory — no matter where your current data directory is positioned within the directory structure. To delete merge from your execution directory just type:

Conclusion

OS9: del -x merge <ENTER>

Most of the time a command uses a small amount of memory called a buffer to perform its work. You can increase the size of the buffer by using a memory modifier. When using copy with large files, you can speed up the process by increasing copy's buffer.

OS9: copy #32k bigfile /d1/BIGFILES/new.bigfile

With no memory modifier, copy transfers around 4000 bytes at a

time. The command line above sets copy's buffer at 32,767 bytes (32K).

There is no replacing the official Tandy manual to OS-9 Level 2. Keep it handy for the full story on the syntax and use for each

Preparing for the Tutorials

More reading than usual.

The last two days you have learned dozens of terms and concepts you will need in the tutorials ahead. These terms and concepts are important, so before you start you might quickly review the Section Summaries.

For these tutorials you should have at hand:

* At least one new box of Double-Sided, Double-Density 5-1/4" disks, including labels and write-protect tabs (if you use a 3-1/2" drive you may need two or three of those also, especially if you only have one 5-1/4" drive).
* A felt-tip pen.
* A list of commonly-called phone numbers (friends, business associates, and so on). You'll be creating a database with these. Optionally, know the addresses and occupations for each person.
* Your Tandy OS-9 Level 2 manual and/or Level II Quick Reference Guide for reference, if you wish.

GETTING STARTED WITH TUTORIAL 1

What we'll cover.

"Getting Started" with Tutorial 1 is lengthy compared to other "Getting Started" sections in this book. The reason? We dis-cuss config, a utility which involves a number of important concepts. It also involves lists of modules requiring explanation.We will step through booting OS-9, backing up your Tandy system disks, and customizing your system using config.

While "Getting Started" is long, Tutorial 1 itself is short. You should find it easy going and sensible, particularly after completing the following material.

Your first job is to read through all of the following material before attempting the tutorial. We will attempt to explain everything completely and in a logical order, but don't be to concerned if you get a little lost. Once the tutorial begins the order and con-tent of the explanations will become clearer. If there is anything you don't quite grasp after the tutorial, simply re-read the ex-planations dealing with that item.

The software purchaser's first job upon returning home is to make backup copies of their new disks. In Tutorial 1 you start off learning how to backup your valuable disks.

The original disks are called distribution masters. After backing them up you should store them away from your computer setup. This keeps fire, water, or — worse yet — YOU from destroying

both a distribution master and its backup. Re-purchasing distribution masters damaged through accident or negligence can quickly lighten a wallet.

Making a backup copy requires you to format a fresh disk to prepare it for use by OS-9. There is a wide variety of possible disk formats. We'll discuss some of those which apply to floppy disk drives.

Once your distribution disks are backed up, it's time to run **config** employing your knowledge of device drivers and descriptors. **Config** creates a custom system disk (within limits). You will also backup this custom system disk.

Commands you will use -- **mfree** *and* **dir**

With this vital preparatory work done, we turn our attention to two important commands. **Mfree** tells you how much memory storage remains unused inside your computer; and **dir** tells you what files and directories reside within the directory you name after dir. If you exclude the directory name parameter and type only: **OS9: dir <ENTER>**
OS-9 shows only the files and directories you have in your current data directory. You may wish to review the introductory reading on data directories before beginning Tutorial 1.

Entering the invisible sector.

When you buy disks from the store they are about as blank as a disk ever gets. Most computers cannot read a disk if it is blank; their operating systems prefer disks with special patterns magnetically applied to them. Your OS-9 CoCo 3 is no exception. Asking an OS-9 computer to read a blank disk (for example, with the dir command) ensures you'll wind up seeing an error number.

Formatting a fresh disk adds the special magnetic patterns to the disk which your OS-9 computer understands. The initial patterns divide the disk into sectors. Some of the sectors, invisible to the dir command, are reserved by OS-9 for certain bookkeeping purposes. You usually don't have or need access to this area. The second area constitutes that disk's root directory.

For example, when you format a 35-track single-sided drive OS-9 reserves 10 sectors for its own use, leaving the remaining 620 sectors for you. The "invisible" sectors contain information about the disk structure. Some of the information housed in these "invisible" sectors includes how the disk is formatted, who created each file, when each file was last updated, and what kind of file each is. OS-9 adds information to the "invisible" sectors

as you add directories, sub-directories, and files to the "visible" sectors. Some OS-9 commands can access this information, allowing you some control over the way OS-9 handles the files on the disk.

The root directory is the topmost directory in a directory system you can easily extend. All other directories created on a formatted disk are placed "below" the root directory. When you boot up, OS-9 sets your current data directory as the root directory of the disk in drive /d0. Tutorial 2 goes into detail on how to change your directory from this root directory to another (hard drive owners, note that your hard drive may be your current data directory on boot-up, and the hard disk's CMDS directory would be the current execution directory).

Disk format options

The amount of information you can store on one disk depends on how the disk is formatted. OS-9's format command offers a wide variety of disk formats. Each format initially reserves the same 10 "invisible" sectors; formats differ mainly in the amount of storage available to you. The format you choose should optimize disk storage within the limits of your disk hardware.

Some of you may be restricted by older Tandy drives which can only be formatted to 35 tracks (or cylinders — 156K). These drives are no longer manufactured. If you have a Tandy FD-501 system you have a single sided 40 track drive (180K). The FD-502 is the only Tandy system to come with 40 track double sided drives (360K). Chances are good if you purchased your drives from a third party supplier (such as OwlWare) that you have 40-track double-sided drives. If you purchased your system used, it is possible that the single sided drives have been replaced with double sided units. Some 5-1/4" drives are 80 track double sided (720K). These have proven popular with many CoCo OS-9ers because they can also read and write 40 track disks. 3-1/2" drives are also used ocassionally, but only the low density type (720K — electrically identical to the 5-1/4" 80 track drives) can be used with standard CoCo disk controllers.

Only two aftermarket supplier, Hemphill Electronics and Frank Hogg Labs, ever made disk controllers that would utilize high density 5-1/4" (1.2MB) and 3-1/2" (1.4MB) floppies for the CoCo, and these systems are rare today. Some OS-9ers have modified the older 12 volt controllers to interface with high density drives. The modifications are extensive and the controller requires a source of +12V for power (not supplied from the CoCo 2 or 3 — a Multi-Pak Interface would be required). Since

most OS-9 files aren't really huge, this modification has not been real popular. If you have a used system, ask the previous owner about the drive capacities.

Six hundred and thirty sectors — the storage space available on a 35-track single-sided drive — stores the equivalent of up to 78 pages of double-spaced text on a disk. Not all of this storage space is directly usable by you since some disk storage is always reserved for OS-9. Also some of it is wasted when a sector, allocated for use, is not completely filled.

A 40-track double-sided drive can hold 1,440 sectors or almost 180 pages of double-spaced text — all on the same sized floppy disk used for 35-track single-sided disks! If you still have the older 35-track single-sided drives, take note: double-sided 40-track or 80-track drives can "pay for themselves" through fewer floppy disk purchases.

Eighty-track double-sided drives store twice what forty-trackers can store. A!most 360 pages of double-spaced text on one floppy disk is astounding technology.

Formatting: Beware the absent modules

The format command can only create disk formats based on device descriptors and drivers in memory. The device descriptors and drivers you use should match the maximum capacity of your drives. With these modules in memory you can format disks to any standard format WHICH DOES NOT EXCEED THE STORAGE ALLOWED BY THE DESCRIPTORS AND DRIVERS. The driver and descriptor modules must be present in the module directory before any formatting can take place. OS-9 can't format a disk in a drive which has no "description".

When you first boot up with the distribution OS-9 Level 2 System Master, the only descriptors and drivers in memory are for 35-track single-sided drives. How, then, can 40-track drive owners format 40-track disks in order to capitalize on their added storage power? The answer: create a new bootfile with the correct descriptors and drivers! We show you how in this tutorial.

How to use **format**

Once the correct descriptors and drivers have been installed on your system, using the format command is easy. Type:

OS9: format /drivename <ENTER>

Note well, however, that the format comand writes over any information on an existing disk. If the disk you format has data on it already, it will be lost. Make sure to DOUBLE-CHECK that a disk you are about to format contains no valuable data.

Consider a common use for format. If drive /d1 contains an expendable disk (either blank or containing data you no longer need), you would type: **OS9: format /d1 <ENTER>**
OS-9 depends on information in the device driver and descriptor for drive /d1 to proceed with the formatting process. Drive /d1 will now be formatted to the maximum capacity allowed in its descriptor.

One "Mastering OS-9" customer owns a 35-track single-sided drive /d0, two 40-track double-sided drives (named /d1 and /d2) and an 80-track double-sided drive built into her top-notch Frank Hogg Labs hard disk. These last two are named /d3 and /h0. Even with these widely varying formats, to format a disk all she has to do is use a system disk with the right descriptors and drivers. The format command takes care of the rest!

Choosing the right format for you

What if the customer mentioned above wanted to place a disk in drive /d1 (a 40-track double-sided drive) and format it for 35-track single-sided operation? Format offers options to allow this. Take a look at the general syntax for format:

OS9: format <drive> <options> <ENTER>

Options are just that — optional. You must give them in the following order, however:

Option	Description
r	Ready to proceed, no need to prompt user
"name"	Give the disk the name in quotes
1 or 2	Number of sides to format
'35', '40', or '80'	Format number of tracks (cylinders) in single quotes

Note the single-quotation marks surrounding the number of tracks. If you want to format a disk to be 35-track single-sided (consistent with the standard Tandy format) use this command syntax: **OS9: format /d1 1 '35' <ENTER>**

Here we assume your fresh disk is in drive /d1 and that you have installed drivers and descriptors in the boot capable of formatting AT LEAST 35-track single sided drives. The numbers one and 35 tell OS-9 to format a single-sided, 35-track disk. What would be the syntax for a double-sided, forty-track disk format? **OS9: format /d1 2 '40' <ENTER>**

Once again, drivers and descriptors must be present in memory

for AT LEAST 40-track double-sided drives. If no options are specified (OS9: format <ENTER>) the disk will be formatted at the maximum capacity the drivers and descriptors allow.

Backing up disks — legal and therapeutic

Most software companies only sell you the disk and the manual while reserving rights to the software itself. Most often, the original software products are legally usable on only one computer. Making backup copies of distributed software for use on a single computer is legal. It also calms the minds of consumers who worry about damaging their distribution copies.

Many software companies in the past have made disk backups impossible through copy-protection. This practice has become rarer to the benefit of consumers. On the other hand, the software industry depends on our honesty. Please do not make illegal backup copies of your software to give to friends. Programmers are not rich. Despite the love many of them have for programming, they do not love struggling to put food on the table.

Distribution masters aren't the only investment to protect through backups. You should also make frequent backups of your data disks to protect hours, weeks, even months of hard work (recall that data disks usually do not contain programs or commands but the fruit of your labor — letters, spreadsheet data, addresses, etc.). Save your sanity — back up your disks.

In this tutorial we backup these disks:
* the Tandy OS9Boot disk (labeled "System Master")
* the Tandy config disk (labeled "Boot/config/Basic 09")
* your "Mastering OS-9" disk.

You will create other disks during the course of the tutorials. Make sure you backup each of these, too.

How to **backup** *disks*

The backup command transfers data from the source disk to the destination disk. The source disk is the disk you wish to back up. The destination disk is the disk onto which the data will be transferred.

NOTE: The destination disk need not be empty as long its format is identical to that of the source disk. All existing data on a destination disk, however, will be lost in the backup process. Since this would create disaster if the destination disk contained valuable information, the backup process asks you if it is OK to wipe out or "scratch" the contents of the destination disk. Because source and destination disks must be identically formatted

for backup to work, you must use other OS-9 commands to transfer data between differently formatted disks. These commands are discussed in later tutorials and include dsave.

Once you determine the disk format that maximizes your disk storage space and install corresponding drivers and descriptors, most of your disks will be identically formatted. As a result, all your backups will be easy.

Place the disk you want to backup in one drive — typically drive /d0 — and place the identically-formatted destination disk in another drive — typically drive /d1. Then type:
OS9: backup <source> <destination> <ENTER>

Most users think of drive /d0 and drive /d1 as the source and destination drives, respectively. To save keystrokes, OS-9 sets up these two drives as the standard assignment for backups. To demonstrate, if you type: **OS9: backup <ENTER>**
your system responds: "Ready to backup from /d0 to /d1 ?:"
Usually just type "y" (yes) to continue. Cautious users will double check that the drives contain the right disks before proceeding. Once the process has begun it is too late to stop without risking damaging BOTH disks. The way backup works may save your data.

A typical session with Backup might go like this:

You type:	**OS9:backup /d0 /d1 <ENTER>**
you see:	**Ready to backup from /d0 to /d1 ?:**
you type:	**< y >**
you see:	**Disk Name**
	(whatever name you gave the disk when formatting)
	is being scratched
	Ok ?:
you type:	**< y >**
you see:	**Sectors copied: $05A0**
	(on 40-track double-sided drives)
	Verify pass
	Sectors verified: $05A0
	OS9:

Notice how carefully OS-9 guards you from using backup unless you are really sure. By forcing you to interact with it, the backup command will likely save you from destroying valuable data one day. Notice also that the number of sectors copied isn't 630, 1440, or 2880. Instead, that "number" ($05AO) has a LETTER in it!
The number $05AO is a hexadecimal (or simply hex) number. If

you had sixteen fingers instead of ten you would very likely count in hex. You can usually tell when a veteran computer user writes hex numbers since they usually precede the hex number by a dollar sign ("$"). There are replacement utilities that give decimal rather than hexadecimal results available. You may wish to consider them after learning to use the supplied hex versions. The essay "Do I Really Have To Learn Hexadecimal?", located at the end of this week's tutorial, explains more about hexadecimal numbers.

In the sample output above, $05AO sectors equals 1,440 sectors in more common decimal notation. Backup has just successfully transferred data from one 40-track double-sided disk to another.

Config *—what it does*

A system disk must contain an OS9Boot file and Kernel, carefully placed so that the Disk Extended Color Basic (DECB) command DOS can find them and begin the boot process. In addition to OS9Boot, OS-9 Level 2 requires two commands in the boot disk's CMDS directory to complete the boot process — grfdrv and Shell. Actually, this is true ONLY if you use windows. Since a productive OS-9 Level 2 system uses windows heavily, we include both grfdrv and Shell.

Config can create a boot with these features for you. With config you can do some "customizing" of your boot disk including individually choosing device drivers, descriptors and commands. It is reasonably automatic.

Before you use config let's review the steps it takes you through. These eleven steps each require an explanation. Wading through this material will pay off when learning to use other OS-9 boot-making utilities.

Config takes these actions:
1) It displays a copyright notice.
2) It prompts you for the number of drives on your system.
3) For systems with two or more drives (as is the case with most users of "Mastering OS-9"), it asks for the name of the source and the destination drive.
4) The screen clears; config then announces that it is building a descriptor list. Descriptors are modules which tell OS-9 the name, location, and "starting values" for a device attached to your CoCo. Config builds this descriptor list so that it can present you with device choices.

5) The first part of the descriptor list appears on your screen.

This part of the list includes device descriptor names for each of the following devices when using the standard Tandy config disk:

Descriptor Name	Device it Describes
P	serial printer
T1	an eternal terminal (keyboard/screen)
T2	another external terminal
T3	another external terminal
M1	Modem-pak for telecommunications (now obsolete!)
M2	another modem-pak
PIPE	Pipe — internal buffer for data transfer
D0_35S	35-track single-sided descriptor for /d0
D1_ 35S	35-track single-sided descriptor for /d1
D2_35S	35-track single-sided descriptor for /d2

If the descriptor for your particular drive, say a 40-track double sided unit, is not present don't worry. There are more descriptors available, but we will go over the ones on screen first.

Move up and down along this list with the arrow keys. Select thedescriptor you want (and therefore the DEVICE you want) by pressing "S". An "X" will appear next to a selected descriptor. If you make a mistake by selecting a wrong descriptor simply press "S" again. The "S" will now switch OFF the "X". Let's examine the list above one line at a time:

P - The P descriptor is required if you have a printer attached to the serial I/0 port on the back of your CoCo. Even if you don't have a printer yet, select P. That way when you do get a prionter your system will already be set up for one (and you will eventually want a printer!)

T1 - T3 - You may be surprised to see no less than three descriptors for external terminals. Remember that OS-9 is multi-user as well as multi-tasking. It is possible to attach three keyboard/monitor combinations to your CoCo and have three more people access the computer's resources while you do your work. Three terminals isnot the limit; however, adding a terminal may slow down your own work, especially under heavy computation and disk access.

M1, M2 - The ModemPak from Tandy is a ROM pak which allows you to call up information services. The ModemPak has a built-in RS-232 port and a 300-baud modem. Needless to say, it

has long been obsolete.

A better choice is a Tandy RS-232 Pak. These are discontinued and require luck to find. The best bet is to purchae an RS-32 Pak work-alike from CoNect (address in appendices).Sometimes called hardware serial ports, they allow you to use up to 14.4K baud modems if you wish, speed1ng up flle transfer rates between you and an electronic bulletin board system (BBS). They can even be used to run an external terminal at up to 19.2K baud. There are plans circulating on CoCo BBS systems to convert the ModemPak to a limited RS-232 serial port. The conversion is useful for using a modem up to 9600 baud, but does not support hardware handshaking required for higher speeds.

PIPE - This device descriptor is associated with the virtual device inside your CoCo called a pipe. As explained in the introductory reading, OS-9 utilities can pass data to each other through such a pipe.

Dx_35S - 35-track single-sided drive descriptors (35 = number of tracks, S = single sided). The original drive system sold by Tandy for the CoCo had a single sided 35 track drive. Luckily, these are becoming rare since it is cheaper to replace a worn or troublesome unit with a higher capacity 40 trak double sided drive. Still, the descriptors and their drivers are included as a courtesy and for backward compatibility.

6) At this point in the config process, press the right arrow key to proceed to the second part of the descriptor list. This part of the list includes these device descriptor names:

Descriptor Name	Device it Describes
D3_35S	35-track single-sided descriptor for /d3
DDD0_35S	35-track single-sided descriptor for default drive
D0_40D	40-track double-sided descriptor for /d0
D1_40D	40-track double-sided descriptor for /d1
D2_40D	40-track double-sided descriptor for /d2
DDD_40D	40-track double-sided descriptor for default drive
D1_80D	80-track double-sided descriptor for /d1
D2_80D	80-track double-sided descriptor for /d2
R0	Descriptor for surprise software from Kevin Darling
HWP	Public-domain descriptor from William Brady

The second list includes descriptor names for 40 and 80 track double-sided drives. It also includes default drive descriptors for 35 track single sided and 40 track double sided drives. Note that there are no default drive descriptors for 80 track double sided drives. OS-9 supports these, and they can be used as default drives by modifying the default descriptors with dmode (a public domain utility). The reason there is no default drive descriptor is that a 35 or 40 track drive is needed for backwards compatibility with other systems and required to boot from the original distribution disk.

Notice the last two descriptors. These modules and their associated device drivers will be "snuck into" your OS-9 bootlist thanks to some "Mastering OS-9" trickery. All we divulge at this point: R0 will dramatically increase your productivity, and MWP and its associated driver will allow you to communicate with electronic BBS's when you have a modem and shareware software such as WizPro. Thanks go to Kevin Darling and Bill Brady for these extras.

Press "S" for each descriptor (with corresponding driver) you wish included in your boot file. Chances are you will have drives capable of more than 35-track single-sided operation. If so, make sure you de-select that descriptor for drive /d0. Choose a descriptor which maximizes drive /d0's storage — D0_40D, for example. Be sure to include a default drive descriptor (DDD_xxx) which matches the format of drive /d0.

An exception to this approach exists for those who own hard drives. For some hard drive systems you must modify or "patch" existing device descriptors. Instructions are usually provided when you purchase a Color Computer hard drive.

Owners of these capacious and incredibly fast drives benefit when their default drive descriptor forces OS-9 to look on the hard drive when accessing the default drive. Programs load quickly and system response is excellent. For more information on hard drives, see Francis Swygert's article in the appendix.

7) The next menu in config asks you to choose your terminal descriptor and windows. In the Tutorial, we have you choose Term_Win over Term_VDG. When you make this selection, you are shown a list of window descriptors labeled W1 through W7. Select all of them for inclusion in your boot file.

8) When window selection is complete, config creates a bootlist which includes only the names of the modules you have chosen. Later, when building the actual OS9Boot, config will refer to the bootlist for guidance.

9) After building the bootlist config prompts you to provide the name of the correct clock module. Two choices are given, 50Hz and 60Hz. American users should choose 60Hz, Australian and most European users 50Hz (refers to cycles per second of the electric power).

10) Now that you have carefully and knowledgeably chosen the descriptors you want in your bootfile, config asks you to insert a freshly-formatted disk in drive /d1. Do so, and the utility proceeds to generate the new bootfile. lt consults its bootlist and, based on the names it finds, tracks down the correct drivers and descriptors from the directory /d0/MODULES.

11) The next and last step for config is to transfer to the (almost) empty disk in drive /d1 any OS-9 commands you wish. For the purposes of these tutorials, transfer a full set of commands to the disk in drive /d1. The Startup file is also transferred to the new disk.

Before commands can be transferred from drive /d0 to drive /d1, the CMDS directory containing the commands must be present in drive /d0. The CMDS directory we seek is on the System disk. You must remove the config disk from drive /d0 and replace it with the System disk when prompted. Then the command transfer can begin.

When you choose to transfer a full set of commands, the Startup file from the System Master is also transferred to the new disk. Startup is not a command, but a special file which OS-9 uses at bootup to tidy and customize your computer operations. The command transfer process takes several minutes.

If you use config to create a system disk which includes ONLY the modules distributed on the config disk, you will probably succeed in making a bootable custom system disk. Problems arise when you use config to add a large number of modules to the boot. Also, sometimes third-party (non-Tandy vendor) modules should be added to the boot with great care.

IF MODULES ARE INSTALLED IN A BOOT IN THE WRONG ORDER, MEMORY ALLOCATION FOR CERTAIN OPERA-

TIONS MAY BE IMPOSSIBLE. The result may be error messages or even unbootable disks. Some users report unreliable performance only on occasion, making the source of the problem hard to track down. OS-9 Level 2's sensitivity to bootlist ordering has been called the "Boot List Order Bug" (BLOB). To learn more about the BLOB, see the appendices.

About **mfree**

There may come a time when you see ERROR #207 or ERROR #237. Sometimes this means your computer's memory is full. Other times, you may be asking OS-9 to do work which requires more than 64K of memory at once — an impossibility resulting from your computer's hardware. Thus, just because you have memory free does not mean you have enough for all tasks.

mfree — *How it works*

To see how much memory you have available for programs, you can type: **OS9: mfree <ENTER>**
You should receive a report such as:

```
Blk Begin  End  Blks  Size
-- ---- ---- ---- ----
 17 2E000 6BFFF  1F   248k
                ==== ======
          Total: 1F   248k
```

The mfree command also gives information about which memory blocks are in use. Most users don't need this information. These blocks, however, will be important to understand when we explore memory management in a later tutorial.

The **dir** *command tells you where you stand*

As mentioned in the introductory reading, OS-9 employs current data directories and current execution directories. At all times, OS-9 must know which is which.

When you boot OS-9 the system software automatically establishes /dd as the current data directory and /dd/CMDS as the current execution directory. To make full use of the directory system you will want to create directories on disk and then move among them with Chd and Chx. We'll discuss and use these commands extensively in Tutorial 2.

Once you have established a current data directory, you may wish to determine which files it contains. A list of these files, called a directory listing, scrolls down your screen when you type: **OS9: dir <ENTER>**

The files in your current data directory will be listed to your screen. For example, here is what I have in my current data directory:

```
  Directory of . 04:19:08
saf1    Tut1.Pt1.dsa  FN_1    Tut1.Pt1.dsa.SCR
```

The dot (".") following "Directory of" tells me that the directory listing is of the current data directory. Thus, the listing hows local file names as defined in the introductory reading.

In the example above there are four files — an original copy of this tutorial ("Tut1.Pt1.dsa"), a "scratch" copy ("Tut1.Pt1.dsa.SCR") used by my word processor, and my own "safety" copy ("saf1"). There is also a file containing a footnote ("FN_1"). The directory containing the above four files is named /d1/TUT1/DAY_THR. Let's examine this pathlist.:

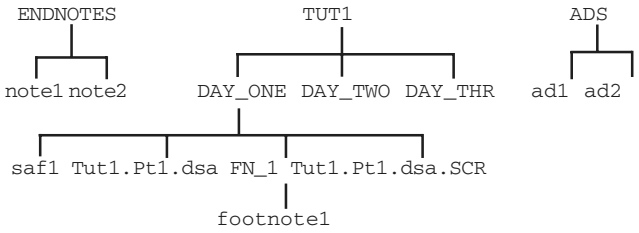
* Drive /d1 is a device (note the slash) for mass data storage; "/d1" refers to the root directory of the disk inside drive /d1.
* "TUT1" is a directory I created beneath the root directory. I use it to store all my first week's tutorials.
* "DAY_THR" is contained within "TUT1" and houses all files related to the third day of the first week's tutorials.

Changing where you stand

Other directories exist on drive /d1. I created separate data directories for each day of the first week, for example. I also have directories for ENDNOTES and ADS. These two directories are placed beneath the root directory along with TUT1. With all these directories and there cumbersome pathnames, chd is handy and saves a lot of typing. When I chd to the root directory of drive /d1 a directory listing reveals:

```
  Directory of . 04:33:30
ENDNOTES   TUT1  ADS
```

The complete directory tree looks like this:



Get ready

This is an action packed tutorial. The important points to note are that it is easy to understand config when you know what it's doing; that dir tells you what files and directories are contained in your current directory; and that you can discover how much memory is available at any time with the mfree command. Keep in mind that safety copies (backups) of your disks are essential!

Tutorial One Notes

NOTES:

1) There ARE magnetic patterns on blank disks. Disk manufacturers test their disks before they leave the factory by writing test patterns on them. But these patterns aren't intelligihle to any personal computer I know of, including the CoCo. So "blank" disks aren't really blank — but they might as well be.

2) The number of tracks affects how much information can be stored, but sometimes the interleave (or skew) of the cylinders strongly affects how quickly this data can be reached, especially with hard disks. Interleave refers to the placement of sectors on each cylinder. An interleave value of one (1) means that sectors are numbered consecutively. A value of five (5) means that there are five sectors skipped between sector one and two. This is done so that the computer is ready to read the next sector by the time it actually arrives under the drive head. An interleave of one on a slow drive means that the disk will have to complete a revolution to before it is ready to read the next sector, thus slowing reads (and writes) down consdierably when compared to a drive formatted with an interleave of five on the same machine. The drive with the interleave of five is faster because the drive heads are over each sector when the computer is ready to read or write that sector.

Format allows you to alter interleaev, but you must be careful. If the computer is capable of reading an interleave of five but the disk was formatted with an interleave of six, it will only be slightly slower. If the disk is formatted with a lower value than the computer is capable of reading, say four in this example, a complete revolution will have to be made to read each sector and the machine will be considerably slower. It is best to leave the floppy drive interleave at the default value of three.

3) You can speed up a multi-user CoCo system in many ways. Get a hard drive, for example. These allow super-fast access to all data without halting the CPU as floppies typically do. Several users can easily use such a CoCo multi-user configuration.

For the money-conscious, consider ramdisks and a no-halt floppy controller. Check with magazines, clubs, on-line services, and CoCo supporting BBSes for used controllers. FARNA Systems still carries a SCSI hard drive interface for the CoCo.

Tutorial 1
Step 1: Booting OS-9

TUTORIAL 1
STEP 1: BOOTING OS-9

1.1 Turn on your system in the following order: Drives, MultiPak
Interface (if any), CoCo 3, monitor. Make sure your drives are empty and all computer system components are correctly connected. If all items are plugged into a switched power strip, they may all be turned on at the same time with the strip switch.

1.2 Find the disk marked "OS9 Level Two Operating System — System Master". Place a write-protect tab over the rectangular notch near the corner of the disk.

1.2 Place the System Master disk in drive /d0 and close the drive door.

1.3 At the BASIC "OK" prompt, type "DOS" followed by the <ENTER> key: **OK: DOS <ENTER>**

1.4 After a moment your screen will show the OS-9 bootup message in the center: **OS9 BOOT**

If you run into any problems, confirm that your system is correctly configured by consulting your computer manuals and try again. Early disk controllers (the type that require 12V from an MPI) do not support the 2MHz clock speed of the CoCo 3 (see Marty Goodman's article, Appendix B). OS-9 Level II automatically boots at 2MHz and cannot be slowed to 1MHz.

1.5 You will see:

```
OS-9 LEVEL TWO VR. 02.00.01
     COPYRIGHT 1986 BY
  MICROWARE SYSTEMS CORP.
  LICENSED TO TANDY CORP.
    ALL RIGHTS RESERVED.
* Welcome to OS-9 LEVEL 2 *
* on the Color Computer 3 *

     yy/mm/dd hh:mm:ss
Time ?
```

1.6 At the Time ? prompt, enter the date and time as indicated (yy:mm:dd hh:mm:ss — year, month, day, hour, minutes, and seconds). One may separate the individual components of time with a space, colon, slash, or not at all. OS-9 will read 95:03:04 14:10:20 and 950304141020 as the same date and time (1995, March 4, 2:10 pm, 20 seconds after). All entries must be filled in order with the exception of seconds. If there is no value for seconds, "00" is assumed (most users do not provide seconds). If no date and time are given, OS-9 sets all values to "00". End the date entry by pressing the <ENTER> key. Note that systems equipped with a real-time clock and the appropriate drivers will automatically display the date and time according to the clock.

Step 2: Formatting Disks

STEP 2: FORMATTING DISKS

2.1 When the boot process finishes you will see Shell and OS9: on the monitor screen. At the "OS9:" prompt, type:

OS9: load format <ENTER>

This loads the format command into memory from the system disk. We'll explore the vices and virtues of in-memory commands in another tutorial. The word "Shell" simply reminds you that the OS-9 Shell is up and running, waiting for a command from you.

2.2 Place a fresh, blank disk in drive /d1. Then type:

OS9: format /d1 <ENTER>

2.3 The screen shows:

```
COLOR COMPTER FORMATTER
Formatting drive /d1
y (yes) or n (no)
Ready ?
```

2.3 The format command requests approval to format the disk in drive /d1. Answer "yes" by pressing <y>.

2.4 Give the disk the name "x" when prompted.

Each time you format a disk, the disk is imprinted with sectors where data can be stored. You also give the disk a name. The name can be anything up to 32 characters including dots. In this case, you could call the disk "System Master Backup" (20 characters including spaces). A name must be provided. The disk name is usually unimportant, so a single space may be used to effectively leave the name blank. The only way to change the name later is to reformat or use a special utility (none is pro-

vided by Tandy).
After the sectors have been created OS-9 checks to make sure data can be written to and read from each sector. This process is called "verification". As each sector is verified, its number is printed to your screen in hexadecimal notation.

2.5 You should see the following report:

```
Number of good sectors: $000276
```

This is the number of sectors (630) in hexadecimal notation. If any sectors were bad, you will get a different number. Since this is your System Master (and the Boot/Config/Basic09 disk later), a disk with bad sectors should not be used. Often formatting a second time will clear any bad sectors (go back to step 2.2.0). If after formatting the same disk a second time there are still bad sectors reported, get another disk. The disk with bad sectors can still be used, since OS-9 will not write on the bad sectors, but it will hold less data and could be less reliable in the future. There is no way to know why the sectors are bad or if the entire disk will be affected later.

2.6 Since we will be backing up the Boot/Config/Basic09 and the Mastering OS-9 disks also, go back to step 2.2.0 and format two more disks now. We will also need a formatted disk to make a custom boot on, so make that three disks to be formatted. If you wish, name these disks "Boot/Config/Basic09 Backup", "Mastering OS-9 Backup", and "Custom Boot 1" instead of "x" or " ". Label the disks accordingly. Remember: don't write on a disk label already on a 5-1/4" disk unless using a felt-tip pen. The pressure from a ball-point pen or pencil could damage the disk. It is always best to write on the label BEFORE placing it on the disk. It would not be a bad idea to go ahead and format two or three additional disks after backing up the Tandy and Mastering OS-9 disks just to have them handy later. One good thing about the OS-9 windowing system is that you can switch to another window and format disks if needed without leaving your application or program. Unless you are using a no-halt floppy controller, however, you will have to wait until format is finished before returning to your work.

Step 3: Backing up the System Master disk

STEP 3: BACKING UP THE SYSTEM MASTER

3.1 When you have completed formatting three disks, place the disk labeled "System Master Backup" in drive /d1 and type:

OS9: unlink format <ENTER> (optional)
OS9: load backup <ENTER>
OS9: backup #56K <ENTER>

3.2 Answer all prompts with "yes" by pressing < y >.

3.3 The backup process begins. When all files have been copied, backup begins a verify pass. When verification is completed, the screen should show:

```
Sectors copied: $0276
Verify pass
Sectors verified: $0276
```

If all sectors do not verify, repeat the process. If they do not verify the second time, get another formatted disk and try again.

Unlink removes the following command name (in this case format) from memory, freeing it for other use. If you intend to format more disks after making the system backups, you may want to leave format in memory for now.

After backup has loaded into memory, we executed it with the command "backup #56K". This increases the size of the memory buffer backup uses to copy all the files from one disk to another. If we had not used a modifier and simply typed "backup <ENTER>", the buffer would have only been 4600 bytes (approximately 4.5K) instead of 56,000. The larger the buffer, the fewer reads and writes. 56K is the largest buffer allowed. This is especially important when doing a single drive backup. In the future, if backup returns an error message, try a smaller buffer or unlink some commands from memory.

One can backup using a single drive by using only one drive identifier ("backup /dx <ENTER>", where /dx is the number of the drive to use). Note that the Tandy manual does not indicate that single drive backups are allowed. The only occasion to use a single drive for backup is when an exact duplicate of a disk is needed and only one drive of the required size is available.

3.4 Remove the System Master Backup disk from drive /d1.

3.5 Place the disk in a sleeve and set it aside.

Another disk-handling tip: Avoid magnetic fields. Don't place disks too close to your monitor (especially the left side — that's usually where the flyback transformer is) or disk drives. Don't place them on or too near a loudspeaker. And don't do what a college student reportedly did: stick them to a refrigerator door with a magnet!

Step 4: Backing up the Mastering OS-9 disk

STEP 4: BACKING UP THE "MASTERING OS-9" DISTRIBUTION DISK

4.1 Place the blank formatted disk labeled "Mastering OS-9 Backup" in drive /d1.

4.2 Remove the System Master disk from drive /d0. Put it in its sleeve and put it away .You won't need the distribution System Master disk again unless the backup made in Step 3 fails for some reason.

4.3 Place the distribution copy of the "Mastering OS-9" disk in drive /d0.

4.4 Type: **OS9: chd /d0 <ENTER>**
This is your way of reminding OS-9 that you have placed a new disk in drive /d0. This tells OS-9 to "change your current data directory to the root directory of the disk in drive /d0".

4.5 Type: **OS9: dir <ENTER>** You will see:

```
Directory of /d1 18:52:04
CMDS     CMDS.S09   ERRATA     HANDY_PROCS
MODULES  SOURCE     TUT2         make_7
make_80  add.mods   test.pipe  TUT6.EDIT.FILES
make_1_2
```

Hmm... here are a few interesting directory and file names. You'll explore each of these during the ten tutorials ahead.

4.6 Type: **OS9: backup #56k <ENTER>**

4.7 Answer all prompts with "yes" by pressing <y> .

4.8 When the backup is complete, remove the disk from drive /d1
and place the new label on it.

4.9 Remove the "Mastering OS-9" distribution disk and store it away in a sleeve.

Step 5: Backing up Boot/Config/Dasic09

STEP 5: BACKING UP THE BOOT/CONFIG/BASIC09 DISK

5.1 Place a write-protect tab over the notch on the disk labeled "OS-9 Level Two Operating System — Boot/config/Basic 09".

5.2 Place the disk in drive /d0.

5.3 Place the blank formatted disk labeled "Boot/Config/Basic09 Backup" in drive /d1.

5.4 Type: **OS9: backup #56k <ENTER>**

5.5 Answer all prompts with <y> to proceed with the backup.

5.6 Put the distribution disk in a sleeve and store it away.

5.7 Place the "Mastering OS-9 Backup" disk in drive /d0 and the "Boot/Config/BasicO9 Backup" disk in Drive /d1.

5.8 Type: **OS9: chd /d0; chx /d0/cmds <ENTER>**
Tip: You can place two or more commands on the same line by separating them with a semi-colon.

5.9 Type: **OS9: add.mods <ENTER>**
The drive lights will come on. If you receive an error message, make sure the correct disks are placed in the drives. The procedure file "add.mods" adds some additional modules from the "Mastering OS-9 Backup" disk to the "Boot/Config/Basic09 Backup" disk.

5.10 Remove both disks from the drives when disk activity stops.

Step 6: Configuring a custom boot w/ **config**

STEP 6: CONFIGURING A CUSTOM DISK WITH CONFIG

6.1 Place the"Boot/Config/Basic09 Backup" disk in drive /d0.

6.2 Place the blank formatted disk labeled "Custom Boot 1" in drive /d1

6.3 Our next step is to tell OS-9 that we have yet another disk in drive/d0. Once again (as in Step 4.5.0), use the chd command:

OS9: chd /d0 <ENTER>

The drive light for drive /d0 should come on for a moment while

OS-9 gets its bearings.

Now that OS-9 knows where its data directory is, we need to tell it where the new execution directory is. Use the chx command for this: **OS9: chx /d0/cmds <ENTER>**

6.4 Type: **OS9: config <ENTER>**
The following copyright message and prompt is displayed:

```
                    CONFIG
            CONFIG VERSION 2.0
            COPYRIGHT 1986 BY
          MICROWARE SYSTEMS CORP.
          REPRODUCED UNDER LICENSE
                TO TANDY CORP.
            ALL RIGHTS RESERVED

        HOW MANY DRIVES DO YOU HAVE:
          1 - ONE DRIVE ONLY
          2 - TWO OR MORE DRIVES
            SELECTION [1,2]
```

For the purposes of "Mastering OS-9 ", you need two disk drives. Actually, for OS-9 in general, two disk drives are wise. While it is possible to operate with only one drive, it is much more pleasurable (and less frustrating!)with two.

6.5 Answer "2" by pressing <2> .

6.6 Next config presents this prompt:

```
        ENTER NAME OF SOURCE DISK:
```

6.7 Tell config that the source drive is drive /d0 by typing: **/d0**
Although config asks for a name, what it really wants is the device number!

6.8 Config will print to your screen:

```
        ENTER NAME OF DEST. DISK:
```

6.9 The destination drive is drive /d1. Type: **/d1**

6.10 Config begins building a descriptor list. You see on your screen:

```
      BUILDING DESCRIPTOR LIST
       . . . . PLEASE WAIT
```

6.11 The first page of the descriptor list appears:

```
 ARROWS - UP/DOWN/MORE/BACK
S - SEL/UNSEL  H - HELP D - DONE

P
T1
T2
T3
M1
M2
PIPE
D0_35S                    X
D1_35S
D2_35S
```

6.12 Use the arrow keys and the <S> key to select the modules needed for your system.

You will probably NOT select Tl, T2, T3, Ml or M2. Nor are you likely to select descriptors for the 35-track single-sided drives — unless all you own is two older 35-track single-sided drives.

Most readers should have forty-track double-sided drives, and a few of you might have 80-track double sided drives (5-1/4" or 3-1/2"). The rest of this tutorial assumes you have selected the forty-track double-sided drive descriptors. If you don't have two forty-trackers, no need to worry. Just keep in mind that your screen may occasionally show something slightly different than what you see printed here.

You should see the following after selecting the drivers for our assumed configuration (two 40-track double sided drives). We will select the printer (P) and PIPE while deselecting the 35 track single-sided drive (D0_35S):

```
ARROWS - UP/DOWN/MORE/BACK
S - SEL/UNSEL  H - HELP D - DONE
P                 X
T1
T2
T3
M1
M2
PIPE              X
D0_35S
D1_35S
```

```
D2_35S
```

6.13 Press the right-arrow key to proceed to the next page of descriptors.

6.14 Use the arrow keys and the < S > key to elect the following descriptors (do NOT select MWP or RO at this time):

```
 ARROWS - UP/DOWN/MORE/BACK
S - SEL/UNSEL  H - HELP D - DONE

D3_35S
DDD_35S
D0_40D                   X
D1_40D                  X
D2_40D
DDD_40D                 X
D1_80D
D2_80D
R0
MWP
```

If you are lucky enough to have an 80-track double-sided drive, select a descriptor for that drive. Notice that the default drive descriptors (whose names begin with "DD") can only be 35 or 40 tracks with existing descriptors. Those of you with 80-trackers must choose the descriptors D1_80D and/or D2_80D depending on whether you have one or two 80-trackers on your system.

Incidentally, 80-track drives are becoming increasingly cheap and easy-to-find. Advertised in most computer magazines, you can easily attach them to your system with instructions received from the appendices. Better yet, purchase a modem and join CompuServe, Delphi, or Genie where you can consult OS-9 experts just about any hour of the day. There are also OS-9 groups on Internet and FIDO net.

6.15 Press <d> to show you are done. Press <y> to show you are sure.

6.16 Config then presents the following screen:

```
SELECT TERM DESCRIPTOR
    1 - TERM_VDG
    2 - TERM_WIN
      H - HELP
```

```
SELECTION [1,2]
```

6.17 Press <2> . Term_VDG (the 32 column screen, jokingly called "Very Dumb Green") is only used by a few old games, and most of those are on self-booting disks. For general work and most, if not all, applications, you will want the 40 column Term_Win screen.

6.18 The next screen asks you to select window descriptors. Choose them all. We're not showing you the screen this time because you should be getting the hang of it by now.

6.19 Press <d> when done. Press <y> to confirm.

6.20 The screen then shows:

```
BUILDING BOOT LIST
. . . PLEASE WAIT
```

6.21 Config presents the following screen:

```
WHAT CLOCK MODULE IS NEEDED?
1 - 60 HZ (AMERICAN POWER)
2 - 50 HZ (EUROPEAN POWER)
   SELECTION [1,2]
```

6.22 Most readers of "Mastering OS-9" will choose <1> . Be sure you know what your alternating current rate is. If you are in doubt, ask! This is the electric power cycle. Choosing the wrong one can cause flickering of the screen and the clock not to keep the correct time. The computer counts the highly stable cycles (Hz) of the electric power to time itself.

6.23 The next screen shows:

```
PLACE  FORMATTED DISK IN
         /D1
HIT ANY KEY TO CONTINUE
```

We've thought ahead: the formatted Custom Boot 1 disk is already in drive /d1. Consider what would happen if we had reached this step in the config process without having a formatted disk on hand. We would have had to STOP the configuring process, format a disk,and START ALL OVER! So, make sure you have a formatted disk on hand BEFORE you use config. Another note. Most of the time when a program or utility says "Hit Any Key To Continue", they lie. What they mean is, "Hit

Any Key (Except the <BREAK> key) To Continue". So go ahead and press any key (except the <BREAK> key!).
6.24 The screen will then show:

```
GENERTING A NEW BOOT
. . . PLEASE WAIT
```

Your drives will rattle for around 90 seconds. This is a good time to leaf through "Mastering OS-9". See you in a minute or so.

Step 7: Adding commands

STEP 7: ADDING COMMANDS TO THE NEW BOOT DISK.

7.1 At this point, a new, custom OS9Boot file exists on the disk in drive /d1. Add commands to the disk by responding to the next screen:

```
DO YOU WISH TO ADD
[N]O COMMANDS, STOP NOW
[F]ULL COMMAND SET
[I]NIDIVIDUALLY SELECT
[H] RECEIVE HELP
SELECTION [N,F,I,H]
```

7.1 Press<F>to tranfer a full command set to the new boot disk.

7.2 Config now prompts you to place a system disk in drive /d0. Place the System Disk Backup you created in Step 3.3.0 in /d0. Follow the directions to continue config .This process takes a few minutes. While config is doing its automatic magic, let's discuss the merits of each choice in the menu above.

Leaving the new boot disk with [N]O commands allows you to later create a CMDS directory on it which contains only those commands and applications you want. If you want to create a disk just for word processing, you may wish to transfer just a few standard OS-9 utilities after config is done. Then add the word processor application and various writing style utilities you may have purchased.

[I]NDIVIDUALLY selecting commands may be the choice you make in the future as you become familiar with each OS-9 command. Until you develop a work "style" at your OS-9 computer, however, you won't know which of the almost 50 OS-9 commands you need. Two commands you MUST include to accomplish windowing are Grfdrv and Shell.

For now, we choose the [F]ULL set of commands. In addition to transferring the commands, this option creates a Startup file on the new boot disk. We can (and will) eliminate commands we don't need in later tutorials.

Hmm... the drives are still working, eh? Browse some more through this book if you like. You'll find articles on music, telecomputing, OS-9 related hardware and software, and so on.

Step 8: Testing the disk drives

STEP 8: TESTING THE NEW DISK — DISK DRIVES

8.1 Remove the System Disk Backup from drive /d0 and put it away.

8.2 Remove the newly configured Custom Boot 1 disk from drive /d1 and put it in drive /d0. Close the drive door and press the reset button.

8.3 After a moment you will see the OS-9 bootup message (OS9 BOOT) in the middle of the screen. A 40-column screen is then created.

8.4 Set the time when prompted.

8.5 Check to see if OS-9 has configured your drives correctly. Place a blank floppy disk in drive /d1. Type:

OS9: format /d1 <ENTER>

Since format hasn't been loaded, the command will first load from the new boot disk in drive /d0.

8.6 Proceed with the format. If you chose 40-track double-sided descriptors and drivers, the format will show $05A0 sectors verified. If you chose 80-track double-sided descriptors and drivers, you should see $0B40 sectors verified.

8.7 Remove the newly formatted disk from drive /d1. Label it as you wish to remind you that it has been formatted.

8.8 Format any remaining blank disk on hand.You'll be needing several formatted disks throughout the Tutorials. We'll remind you to format a disk each time you need one, but having a formatted disk already handy will save you time. You may want to load the format command to speed up the formatting process slightly should you decide to format your remaining blank disks

Step 9: Testing pipes

now.

STEP 9: TESTING THE NEW DISK — PIPES

9.1 Place the "Mastering OS-9 Backup" disk in drive /d1. Type:

OS9: chx /d1/cmds <ENTER>

Think through what you have just done. OS-9 will now look for all its commands in memory and then in /d1/CMDS. The CMDS directory on /d1 had better contain any commands you need which are not already in memory!

9.2 Now type: **OS9: chd /d1 <ENTER>**

Now OS-9 will look first in the root directory of the disk in drive /d1 for data.

9.3 Type: **OS9: list test.pipe <ENTER>**

Even though we haven't "formally" met the list command, you can probably guess what it does. In this case, it lists the text file "test.pipe" to your screen.

9.4 You should see:

```
Each
Sensibly
Perfectly
Gets
Steve
Goldberg's
Utility: "Sort"
Re-arranged and
Sorted by
Entry
```

Look carefully. If you read it as a sentence it almost makes sense. This list is really a scrambled statement. Actually, the list is a scrambled commercial for Steve Goldberg's sort utility, included on your "Mastering OS-9" disk.

Including Sort and other excellent Goldberg Utilities with the "Mastering OS-9" disk is our way of introducing OS-9 users to utilities and commands not available on your Level 2 disks. Tandy sells additional utilities with the Development Pak for Level 2. Companies like CoNect, Sub-Etha Software, FARNA Systems, and others offer great packages full of handy commands — at reasonable cost. Check with these and other companies before you buy.

Enough commercial break, let's finish this first tutorial up!

9.5 Type: **OS9: list test.pipe ! sort <ENTER>**
You should see:

```
Each
Entry
Gets
Perfectly
Re-arranged and
Sensibly
Sorted by
Steve Goldberg's
Utility: "Sort"
```

So pipes work too. The ! symbol (exclamation mark) is the pipe modifier. When the Shell encounters this symbol, it sends output of the command preceding it as input to the command following it.

Thus, **list test.pipe** generates as output a listing of the "scrambled" sentence. The pipe transfers this as input to sort which alphabetizes any list that comes its way. Sort's output is the alphabetized list you see above.

Step 10: Testing windows

STEP 10: TESTING THE NEW DISK — WINDOWS

10.1 Type: **OS9: chd /d0 <ENTER>**

10.2 Type: **OS9: chx /d0/cmds <ENTER>**
Now you tell me: where will OS-9 look for all its data? Where will OS-9 look for all its commands (after checking memory, of course)? Note that both commands could have been combined on a single line by typing: OS9: chd /d0;chx /d0/cmds <ENTER>

10.3 Now let's type:
OS9: iniz /w7 <ENTER>
OS9: echo Hello > /w7 <ENTER>
Don't worry about what these commands mean for now. They open windows, a topic covered in a later tutorial.

10.4 If using an RGB monitor such as the Tandy CM-8, Commodore 1084, or Magnavox 8CM515 or 1CM135, type:
OS9: montype r <ENTER>
If using a color composite monitor, type:
OS9: montype c <ENTER>
If using a monochrome monitor, type:

OS9: montype m <ENTER>

These commands make your screen more readable for each particular monitor type. Later, we will add the montype command to our startup file.

10.5 Press the <CLEAR> key. Your screen should change to an 80 column "window". If it did, windows work too! Press the clear key again to return to your original 40 column screen.

I'm getting ahead of myself here, but I can't resist! While in the 40 column screen, type: **OS9: dir** Now press <CLEAR> while the disk drive is running. Type **OS9: mdir <ENTER>** Now press <CLEAR> again. Press <CLEAR> once more. This is the power of a windowing operating system: having more than one application open at once. If you had been working on a program in one window and discovered you needed to format a disk to save your work on, you can merely toggle over to the other window and format a disk *without* quitting and losing any work!

Step 11: Closing down Tutorial 1

STEP 11: CLOSING DOWN SHOP

11.1 Remove all disks from the drives. Place them in sleeves and store them away.

11.2 Switch off system power. If all units are attached to a single power strip, turn off the strip. Otherwise, switch off the monitor, CoCo, MultiPak (if any), and drives in that order.

11.3 Now for some homework! Read up on Tutorial 2 before we get started on it another day!

GETTING STARTED WITH TUTORIAL 2

Tutorial 2
What we'll cover

In Tutorial 2 we take care of business unfinished in Tutorial 1. For example, the custom system disk you created in the first tutorial, while perfectly bootable, omits the SYS directory found on most system disks, including the Tandy distribution disk.

Once we transfer a SYS directory to your new custom disk, we'll back it up. We then explore the directory system on that disk with the chx and chd commands. We'll list a couple of files and use dir and dir x to find the contents of directories.

We also examine the uses for error and help, two commands which make life easier for the first-time user of OS-9 Level 2.

The SYS directory and the startup file

Most OS-9 system disks for the Color Computer include the following in the root directory:

```
 Directory of /d0 11:20:38
OS9Boot  CMDS     SYS      Startup
```

You already know that OS9Boot contains OS-9 modules and that CMDS contains your executable commands, utilities, and applications.

What about the SYS directory? SYS frequently serves as a handy "catch-all" directory containing graphics fonts and patterns, data files which initialize OS-9 applications, error and help messages, and so on. Many OS-9 Level 2 applications are written to seek the default drive's SYS directory, using /dd/SYS in its traditional catch-all role. Without such a directory on your default drive these applications cannot function.

The last common item in a system disk's root directory is the startup file, a "script" telling OS-9 which housekeeping chores to take care of before giving you control of the computer. The startup file can be as simple or as powerful as you wish to make it. All you have to do is add and delete instructions with a text editor.

Typical startup files prompt you to set the system time, start windows automatically, load favorite utilities and modules, and "patch" OS-9 to run better on your hardware. By the time you finish "Mastering OS-9", you will have edited startup to do all of these.
In Tutorial 2 we have you list the Startup file to your screen just to get a broad idea of how Startup works. We leave the details of its role and function for another Tutorial.

dir x — *the other directory*

OS-9 employs two current directories, the current data directory and the current execution directory. You can change which directories OS-9 looks to for data and commands with the chd and chx commands. To find a listing of the contents of the current data directory, type: **OS9: dir <ENTER>** No need to type a full pathname.

What about the execution directory? If /d0/CMDS is your current execution directory (as it most often is) you could type:

OS9: dir /d0/cmds <ENTER>

to find out what commands are available to you on disk. You may need to check the contents of your execution directory often when you first start using OS-9 Level 2.

But typing the above command line (12 keystrokes) can be aggravating. Avoid aggravation. Just type:

OS9: dir x <ENTER>

to see a listing of the contents of the current execution directory. It's easy to remember (the "x" stands for "execution") and it's only five keystrokes. The element of the OS-9 Ethos shown here: if at all possible, avoid unnecessary keystrokes, and make commands easy to remember.

error — *How to not crack a book*

If only you could get help using OS-9 without having that enormous Tandy manual in your lap... no problem! OS-9 helps you up the learning curve by providing useful error messages and on-line help. This is a big improvement over the Microsoft Disk Extended Basic approach. Most user errors simply return ?SN ERROR — not very helpful in determining the precise error.

Under OS-9, if you make a mistake or if OS-9 has painted itself into a corner (see the comments on mfree in Tutorial 1), you receive an error message on your screen. For example, if your fingers slip and you type **OS9: di <ENTER>** instead of **OS9: dir <ENTER>** the shell, unable to find the command "di", will return **ERROR #216** to your screen. ERROR #216 means the shell couldn't find the pathname you specified. To verify this, type: **OS9: error 216 <ENTER>**

At this point the shell goes searching for two items: the error command (which should be in the current execution directory) and the errmsg file in /dd/SYS. If OS-9 succeeds on both counts, you see on your screen:

```
216 - Path Name Not Found
```

Chances are excellent most of your "flubs" will be typographic, so you'll get familiar with error #216. Otherwise, with the grounding in OS-9 you receive from "Mastering OS-9", you may need error to translate error numbers only rarely.

The **help** *command and* **helpmsg** *file*

Another handy file you should tuck into /dd/SYS is helpmsg. This file contains usage and syntax summaries for each OS-9 command. To access a command's summary, type:

OS9: help commandname <ENTER>

Example: To find out information on the cobbler command, type:

OS9: help cobbler <ENTER>

The shell searches for the help command in the current execution
directory; then it searches for information on the cobbler command in /dd/SYS/Helpmsg. If successful, OS-9 then shows the following on your screen:

```
Syntax:Cobbler devname
Usage :Creates 0S-9 bootstrap file from current boot
```

You might guess from the above that creating a boot file on a disk in drive /dl involves typing:

OS9: cobbler /dl <ENTER>

Good guess (this command line works best if the disk in drive /dl is freshly formatted)! You figured it out without cracking open a single book! We cover the cobbler command in greater detail later.

Example: The modpatch command (unmentioned in the Tandy manual) is a handy tool for patching OS-9 — if you know how to use it. To get some help, type:

OS9: help modpatch <ENTER>

```
Syntax:Modpatch <filename> [opts]
Usage :patch a module in memory from command file
Opts  :-s = silent mode
       -w = suppress warnings
      -c = compare module only, do not change
       -? = receive help

Cmds  :L modname = link to module
      C off obyte nbyte=change obyte at
         off(set) to nbyte
       V = verify module
```

```
M = mask IRQs
U = unmask IRQs
```

Getting ready

Look over that last report. True, the report might be more help if you knew what "IRQs", "bytes", and "offsets" were. This information, however, can help the knowledgeable user change OS-9 to access drives faster, alter keyboard repeat delay, and add other streamlining features.We'll be making patches such as these in later tutorials.

Since a typical OS-9 work session requires perhaps 20 commands, don't expect to understand every single line you type during the course of Tutorial 2. Though some command lines may now seem mysterious, by the end of "Mastering OS-9" you will be able to review earlier tutorials and understand every line they contain.

Tutorial 2
Step 1: Transferring SYS and other files

For this tutorial you'll need the "Config/Basic09 Backup", the "System Disk Backup", and "Custom Disk #1". You'll also need the "Mastering OS-9 Backup" disk.

TUTORIAL 2
STEP 1: TRANSFERRING SYS AND HANDY FILES

1.1 Turn on your system.

1.2 Boot up OS-9 with the "System Master Backup", NOT the configured "Custom Boot 1". Set the date/time when prompted.

1.3 Place the "Custom Boot 1" disk in drive /d1.

1.4 Type: **OS9: makdir /d1/SYS <ENTER>**
Make sure you capitalize SYS. When you create directories, always remember to name them with all capital letters.

1.5 Now type: **OS9: chd /d0/sys <ENTER>**
Even though you capitalize a directory name when making it, OS-9 doesn't care what case it is typed in later. It will look for a file or directory in the same case you typed first. If it doesn't find a file or directory exactly as typed, OS-9 will search for any case combination (it would find Sys or SyS also).

1.6 Type: **OS9: dsave /d0 /d1/sys ! shell <ENTER>**
There are spaces between "dsave", "/d0", and "/d1/sys". Don't forget to include them. The exclamation point is the "pipe"

symbol you first saw in Tutorial 1. If you guessed that the output of the dsave command is being sent through a pipe to the shell, fantastic! You are absolutely right.

If you are lost, don't worry. We will explore dsave and pipes more extensively in later tutorials. In the meantime, just watch the scree — OS-9 is automating a tedious job for you. This is what computers are for!

1.7 Type: **OS9: chd /d0 <ENTER>**

1.8 Now type:
OS9: copy window.t80s /d1/window.t80s <ENTER>

1.9 Remove the "System Master Backup" from drive /d0. Put it away in a sleeve.

1.10 Place the "Mastering OS-9 Backup" disk in drive /d0 then type: **OS9: chx /d0/cmds <ENTER>**
Notice that we used the chx command. This tells the shell that if it can't find a command in memory, it should look in the CMDS directory on the disk in drive /d0.

1.11 Type: **OS9: chd /d0 <ENTER>**
Now OS-9 will look for data first in the root directory of the disk in drive/d0 unless you specify another pathname.

1.12 We now need to copy a couple more files by typing:
OS9: copy make_80 /d1/make_80 <ENTER>
OS9: copy make_7 /d1/make_7 <ENTER>
To create the underscore character you see between "make" and "80", hold down the <CTRL> key and press the hyphen (dash) key. This is used a lot in OS-9!

1.13 Remove "Mastering OS-9 Backup" from drive /d0. Keep it close at hand.

1.14 Place "Boot/Config/Basic09 Backup" in drive /d0 then type: **OS9:chd /d0 <ENTER>**
OS9:chx /d0/cmds <ENTER>
Where will the shell now look for commands on disk? Where is the current data directory?

1.15 Now to copy another file. Type:
OS9:copy /d0/cmds/runb /d1/cmds/runb <ENTER>

Step 2: Backing up the custom master

1.16 Remove the "Boot/Config/Basic09 Backup" disk from drive /d0 and put it away.

STEP 2: BACKING UP THE CUSTOM MASTER

2.1 Remove "Custom Master l" from drive /dl and place it in drive /d0.

2.2 Write "Custom Master 1 Backup" on a disk label.

2.3 Place the label on a fresh, blank disk.

2.4 Place the blank disk in drive /d1 and type:

OS9: chd /d0 <ENTER>
OS9: chx /d0/cmds <ENTER>

2.5 Now type: **OS9: format /d1 <ENTER>**
This formats the disk in drive /dl to be a 35-track single-sided disk. Remember, backup cannot proceed if the destination disk's format differs from the source disk's format. Since we booted with the "System Master Backup" (which uses 35-track single-sided descriptors for /d1) we do not need to use format's options. If we had booted with the new "Custom Disk 1", we would have had to type:

OS9: format /d1 1 '35' <ENTER>

2.6 Answer all prompts to proceed with the format. Choose any disk name (or none by pressing the space bar).

2.7 When the formatting has finished, type:
: **OS9: backup #56k <ENTER>**

2.8 Proceed with the backup process.

2.9 When backup is done, remove "Custom Master 1 " from drive /d0, place a write-protect tab over the corner notch, and put it away.

2.10 Remove "Custom Master l Backup" from drive /d1 and place it in drive /d0.

2.12 Reboot OS-9 by pressing the reset button while leaving all disks in place. Set the time. When the"OS9:" prompt appears, proceed to Step 3.

STEP 3: EXPLORING THE DIRECTORY SYSTEM WITH CHX AND CHD

3.1 Type: **OS9: dir <ENTER>**
As promised in "Getting Started with Tutorial 2 ", you can see at least OS9Boot, CMDS, SYS, and startup in the root directory.

3.2 Display a listing of the current execution directory by typing:

OS9: dir x <ENTER>

3.3 Now let's display the contents of the CMDS directory in four ways: **OS9: dir cmds <ENTER>**
OS9: dir CMDS <ENTER>
OS9: dir /d0/cmds <ENTER>
OS9: dir /dd/cmds <ENTER>
You obviously have lots of options to find that listing! In the last line, we assumed drive /d0 is your defauIt drive (/dd). Notice that OS-9 doesn't care which case you use — upper or lower — when reading a directory name. Typing "dir x" is much easier than any of the above four command lines.

3.4 Let's now take a look at the SYS directory, change directories, and then check to make sure OS-9 is where we want to be. Type the following three lines:

OS9: dir sys <ENTER>
OS9: chd sys <ENTER>
OS9: dir <ENTER>

What are these files? Where are you now in the directory system?

/d0

CMDS *SYS* OS9Boot Startup ...

You are now inside the SYS directory, marked on both sides by asterisks in our graph above. The f iles you see are NSIDE the SYS directory.

3.5 Let's take a look at the "errmsg" and "helpmsg" files. Type:
OS9: list errmsg <ENTER>
Press the space bar to continue the listing. When finished viewing the error message file, type:

Step 4: Help with commands and errors

OS9: list helpmsg <ENTER>

Press the space bar to continue the listing. OS-9 uses these two files in conjunction with the error and help commands. Let's explore these now.

STEP 4: GETTING HELP WITH COMMANDS AND ERROR MESSAGES

4.1 Let's cause OS-9 to generate an error message by deliberately misspelling the "dir" command: OS9:di < ENTER >

4.2 Investigate the error message returned by typing:

OS9: error 216 <ENTER>

Leave off the number/pound sign (#).

4.3 Generate another error mesage with:

OS9: dir /d0/cmd <ENTER>

Obviously, "Path Name Not Found" doesn't merely refer to misspelled commands. Just about anything you misspell returns error number 216 — incorrectly typed pathnames too!

4.4 Generate another error message with

OS9: iniz d3 <ENTER>

Iniz takes as a parameter the legal name of a device descriptor. Since we have no device descriptors for a drive named "d3 ", OS-9 sends us an error number.

4.5 Use error to f ind out what the number returned means by typing: **OS9: error 221 <ENTER>**

No surprise here. Of course OS-9 couldn't f ind that module since we didn't include it in our bootlist from Tutorial l.

4.6 Let's move on to the help command. Type:

OS9: help makdir <ENTER>

We used the makdir command in step 1.4. Help tells us makdir's usage and syntax.

4.7 Lets try again: **OS9: help dsave <ENTER>**

This command is obviously a lot more involved than makdir. We'll cover makdir in Tutorial 3 and dsave in Tutorials 7 and 8.

4.8 Okay, one more time: **OS9: help shell <ENTER>**

In Step 1.6, the shell, which offers us the "0S9: " prompt, ran another shell explicitly. Imagine, a shell running a shell — both running at the same time! You started a multi-tasking process and didn't even know it!

STEP 5: INVESTIGATING THE STARTUP FILE

5.1 Change directories to the root directory of drive /d0:

OS9: chd /d0 <ENTER>

5.2 Now let's list the "startup" file to the screen:

OS9: list startup <ENTER>

You will see:

```
* Echo welcome message
echo * Welcone to OS-9 LEVEL 2 *
echo * on the Color Conputer 3 *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
```

Each line is a perfectly acceptable OS-9 command line. To prove it, let's type each line at a prompt:

OS9: * Echo welcome message <ENTER>

This command line does nothing — it doesn't even return an error message! The asterisk at the beginning of the line tips off the shell to ignore what follows, all the way up to the <ENTER>.

OS9: echo * Welcome to OS-9 LEVEL 2 * <ENTER>
(Now THIS does something!)
OS9: echo * on the Color Computer 3 * <ENTER>
OS9: * Lock shell and std utils into memory <ENTER>
OS9: link shell <ENTER>
OS9: * Start system time from keyboard <ENTER>
OS9: setime < /1 <ENTER> *(Familiar?)*
OS9: date t <ENTER> *(You should remember this, too.)*

Now you probably have an inkling about how startup works. Startup is an example of a procedure file.We explore procedure files in detail later.

Step 6: Directories and subdirectories

STEP 6: INVESTIGATING DIRECTORIES AND SUBDIRECTORIES

6.1 Place the "Mastering OS-9 Backup" disk in drive /d1.

6.2 Type the following command lines:

OS9: make_7 <ENTER>
OS9: load pwd runb <ENTER>
OS9: load /d1/cmds/where <ENTER>
OS9: chd /d1/tut2 <ENTER>
OS9: dir <ENTER>

You will see:

```
 Directory of . 12:00:03
BUSINESS CABINET  where
```

Throughout Steps 6 and 7 of this tutorial, each directory and subdirectory will contain a file called "where". Ignore it. Or list it if you wish. It exists for the benefit of the where command I wrote to help you keep track of your directory-system-where-abouts.

6.3 To see where you are in relation to the directory system created under TUT2, type: **OS9: where <ENTER>** and then press the <CLEAR> key. You will see a graphic representation of your location. The directory in which you are located is marked on both sides with an asterisk. Press <CLEAR> again to return to your work (this is your first session with windows, by the way).

6.4 Let's see what's in the CABINET directory (we know it's a directory because we always follow the rule that directories use all capital letters, right?):

OS9: chd cabinet <ENTER>
OS9: dir <ENTER>

Imagine the disk in drive /dl is a house of information. Enter the house and you f ind a room named TUT2. Enter it (with chd) and look around (with dir). Ah, there's a CABINET in this room. Look inside (once again with dir). But which DRAWER do we

investigate?

6.5 Type: **OS9: chd drawerl <ENTER>**
What have we here?

6.6 Now type: **OS9: dir <ENTER>**
A folder! Chances are good it contains some files...

6.7 Let's see what's in FOLDER1:
OS9: chd folderl <ENTER>
OS9: dir <ENTER>
Here you see four f iles. Do you see how well organized you can be with such a directory system? Think of all the information a person handles. Now no one has an excuse for not having a LOGICAL place to put their information. Just create a directory!

Step 7: Applying the directory system

Once again, if you get lost in this directory system, type "where" at the "OS9:" prompt and proceed as instructed in Step 6.5

STEP 7: A PRACTICAL APPLICATION OF THE DIRECTORY SYSTEM

7.1 Let's change directories again:
OS9: chd /d1/tut2 <ENTER>

7.2 Now "where" are we in the directory system?:
OS9: where <ENTER>
Press the <CLEAR> key to check your whereabouts. Press it again to return.

7.3 We'll now take a look at what is in TUT2, change directories once more to BUSINESS, and then see exactly "where" we are again: **OS9: dir <ENTER>**
OS9: chd business <ENTER>
OS9: where <ENTER>
Press the <CLEAR> key. Hopefully your movement around the directory tree makes more and more sense as we do this!

7.4 Let's move around some more then again check our bearings: **OS9: dir <ENTER>**
OS9: chd sales <ENTER>
OS9: dir <ENTER>
OS9: where <ENTER>

You should be getting the hang of this by now. Take your time

wandering through the directory structure you see listed on the 80-column screen. Be sure to go all the way to the bottom of the BUSINESS directory. To climb back up, either go right to the top with:: **OS9: chd /dl/tut2 <ENTER>** or go back up one level at a time by specifying complete pathnames for each directory. Or use anonymous directories!

Step 8: Closing down Tutorial 2

If you run into problems, remember
* You can't use "list" on directories
* You can't use "dir" on files
* You can't chd to files
* "help" and "error" are always waiting for you.

Good luck!

STEP 8: CLOSING DOWN SHOP

8.1 When you are finished exploring the directory tree, remove all disks from the drives.

8.2 Power down the computer, monitor, MPI (if you have one), and drives (or turn all off together if on a power strip).

8.3.0 Put all disks away in their sleeves.

GETTING STARTED WITH TUTORIAL 3

Tutorial 3: What we'll cover

You now get first-hand experience in creating directories. You will also learn how to delete them. We explore file attributes which apply not only to files but to directories as well, pointing out a fundamental similarity between them.You will learn how to rename files and directories. You will also discover yet another kind of directory listing, **dir e**, which provides you with extensive and useful file information.

A file is a file is a file — is a module

Recall that the basic unit of your work under OS-9 is the file. Command files and data files are obvious examples. Directories, too, are files with special attributes distinguishing them from "ordinary" files. A directory file contains file name entries its local files and shorthand entries for itself and its parent directory

To recap:
* Command files contain machine codes which constitute programs, utllities, and commands.
* Data files contain data used and created by programs, utilities, and commands.
* Directories file away names of command files, data files, and other directories (these last names are then subdirectories).

In addition to these three classes of files, another class is available to OS-9: OS-9 modules. These are not usually used as either commands or program data. Examples are the device drivers and device descriptors mentioned in earlier reading.

Describing files

Just as modules are files, files can be viewed as modules. Both are essential units in the OS-9 operating system and in your work. Viewing every "object" on your system as a file allows OS-9 to manipulate them in a similar, "unified" way. lt is part of what makes OS-9 both simple and powerful.

For each file there is a wealth of information on the disk. A directory contains the name of a file and a pointer to a file descriptor. This file descriptor contains a summary of a file's attributes. You can see a file's attribute report by typing:

OS9: attr filename <ENTER>

You can use the attr command to change a file's attributes also. We'll use this handy tool often.

On multi-user systems, one person is in charge of "straightening

More on file attributes

up" the system (deleting unnecessary files, organizing who can use the system and who cannot, and so on). This person, the "superuser", has wider access to files than anyone else with the ability to read and write to most files on the system, no matter who the files belong to.

On OS-9 multi-user systems, the superuser has a special "User Identification Number" of zero. On your own CoCo system, your
User ID Number is zero, making you the superuser. Conversely, users who are not superuser usually have less access to a computer system. This helps protect the system from being damaged through accident or abuse.

A multi-user CoCo 3 Level 2 system is easy toassemble. One Radio Shack franchiser in Delaware runs his store on Wyse terminals hooked up to a CoCo 3 running OS-9 Level 2. It's as easy for you to do. With simple software and hardware serial ports and terminals (or computers running terminal emulation software), you could have a Color Computer terminal in every room of your home (and probably cause domestic problems thereby).

File attributes summarize who is permitted access to files. Some of these attributes tell OS-9 that a file may be read or written to only by the superuser — you. Other attributes tell
OS-9 that the public can read or write to a file. Permission to read or write to a file may even be denied to both you AND the public! The file is thus unusable until its attributes are deliberately set to give you appropriate permission (we explain set and reset attributes later).

When you request an attribute report with attr, the screen shows a series of letters and hyphens representing permissions granted and denied. A quick example is in order:

OS9: attr <directory_name> <ENTER>

may provide the following report:

```
d-rwerwe
```

The position of the letters and hyphens determines precisely which permission has been granted or denied.

Study this table:

Table 3: File Attributes

Position	Character	Permission
1	"d"	access only as a directory
2	"s"	used by single user only
3	"r"	user 0 can read the file
4	"w"	user 0 can write to the file
5	"e"	user 0 can execute the file
6	"r"	public can read the file
7	"w"	public can write to the file
8	"e"	public can execute the file

A dash in the position indicates the permission is NOT granted (i.e. — in the example "d-rwerwe", the file is NOT useable by a single user, so can be accessed by ALL users).

You can see from the above example and table that the directory named "directory_name" is:

1) a directory (no surprises here!)
2) shareable
3) publicly available to be written to, read from, and executed
4) available to you as user 0 to write to, read from, and execute

How OS-9 uses file attributes

We say that a file is a directory if its directory attribute is set. We also say that a file is not a directory if its directory attribute is reset. The same is true for the other attrihutes — if the letters "s","e", "r", and so on appear in the attribute report then these attributes are set. If instead you see hyphens, these attributes are reset. The reason for this language goes back to computer programming at the bit and byte level and is beyond the scope of this book.

OS-9 cares which attributes are set and reset because many system operations and commands will only work on files with specific attributes set. For example:

* You can delete (del) a file, but not a directory.
* You can delete a directory with deldir, but it won't work on files.
* You can copy files, but not directories — at least not with the copy command.
* You can't list a file with its read and public-read attributes reset, nor can you list a directory.
* A directory whose execute and public-execute attributes are reset cannot be used as an execution directory.

Deleting directories

Since the del command does not work on directories, you may wonder how a directory is deleted:
1) One way to delete a directory SAFELY would be to delete every file in the directory with the del command. This is tedious but forces you to reconsider deleting each and every file.

2) Once all files are deleted, you can use the attr command to remove the directory attribute of the directory, making it an ordinary file. You cannot reset a directory's "d" attribufe unless it is empty.

3) Last, delete the "directory" (now just a file) with the del command.

Commands you will use -- **deldir**

Or you can just use the **deldir** command. Deldir works by automating the procedure outlined above.You must use deldir with great care, as it can wipe out many weeks or months of work. Even though you must use EXTREME caution when deleting directories (particularly while you learn the directory system), deldir does provide you the option of examining the contents of each and every directory it eliminates.

When to change a file's attributes

Most of you will be the only user on your OS-9 CoCo so you don't require secret passwords, log-on sequences, and reset attributes to control access to files. When, then, will you need to change a file's attributes? There are chiefly two occasions to use the attr command on a single-user system:

1) To protect files of extreme importance, you can deliberately deny write permissions. Months later you won't unknowingly delete these files with the del or deldir commands.

2) Command files joined together with the merge command cannot be executed. The attr command can be used to grant execution permission.

In Tutorial 3 we use the chd command to move around the directory structure. As we proceed create a mental image of the disk's directory structure.

TUTORIAL 3

STEP 1: CREATING DIRECTORIES

1.0 Boot OS-9 with "Custom Disk 1 Backup"

1.2 Get out "Mastering OS-9 Backup". Remove the write-protect tab if necessary.

1.3 Place "Mastering OS-9 Backup" in drive /d1

1.4 Type: **OS9: chd /dl <ENTER>**
OS9: makdir MY_DIR <ENTER>

Once again, you create the underscore character by holding down
the <CTRL> key while you press the hyphen key. This character is used all the time by old-time OS-9ers, although I'm not sure why (probably a carry over from some programmer's main-frame days).

1.5 Now type: **OS9: dir <ENTER>**

Notice that there is a new directory named MY_DIR. You just created this directory yourself with the makdir command. You capitalized the name, respecting OS-9's convention to use capital letters when naming directories and lowercase letters when naming files. If you didn't do this, it would be impossible to distinguish ordinary files from directories.

1.6 Now let's make a subdirectory:

OS9: makdir my_dir/BUSlNESS <ENTER>

Notice we did not capitalize "my_dir ". OS-9 will find MY_DIR whether you type MY_DIR, My_Dir, my_DIR, or some other combination. OS-9 is NOT case-sensitive when READING file names and directory names; it IS case-sensitive when WRITING file names. This is why you capitalized BUSlNESS; 0S-9 creates the directory with all capital letters in its name because creating that name on disk is a WRITE operation. Remember, a subdirectory is a directory within (or under) another directory.

1.7 Make another subdirectory by typing:

OS9: makdir my dir/PERSONAL <ENTER>

1.8 Get a directory listing of MY_DIR to confirm your work.

1.9 Change your current data directory to my_dir/business and then make two more directories by typing

OS9: chd my_dir/business <ENTER>
OS9: makdir PROPOSALS <ENTER>

OS9: makdir LETTERS <ENTER>

1.10 Get a directory listing.

1.11 Change your current data directory to my dir/business/proposals: **OS9: chd proposals <ENTER>**

Remember that the full pathname of PROPOSALS is really /dl/my_dir/business/proposals. By setting our current data directory to /dl/my_dir/business, we save ourselves from typing the first part of the pathname in the above step.

1.12 We'll now make some subdirectories under PROPOSALS and PERSONAL then inspect our work:

OS9: makdir NEWSLETTER <ENTER>
OS9: makdir COST_CUTS <ENTER>
OS9: dir <ENTER>
OS9: chd /d1/my_dir/personal <ENTER>
OS9: makdir LETTERS <ENTER>
OS9: dir <ENTER>

1.13 Did you follow what happened in step 1.12? If not, go back through the directories using the full path names and chd command. You should be conf ident enough now to create your own directories. Remember: if you want to create a directory in the current directory just use a local directory name after typing makdir. To create directories elsewhere, specify enough of a path name so that OS-9 positions the directory where you want it in the directory system.

Step 2: Working with file attributes

Commands you will use -- **dir e**

STEP 2: WORKING WITH FILE ATTRIBUTES

2.1 We now learn how to view data hidden in these directories, data.such as the date and time each file and directory was last modified and the file attributes of each. This data is revealed by another variant of the dir command — **dir e**.

2.2 Type the following:

OS9: chd /dl/my_dir <ENTER>
OS9: dir e personal <ENTER>

You should receive a report on your screen similar to this:

```
Directory of /dl/my_dir/personal  18:34:44

Owner Last modified Attributes Sector Bytecount Name
——— ———————— —————— ——— ————— ----------
  0  95/03/07 1827  d-ewrewr     F7       40 LETTERS
```

Notice that the owner of the LETTERS directory has number zero. This is the User Identification Number mentioned at the beginning of the tutorial. The next columns give the date and time the directory was last modified, the attribute report (in this case, the file is a directory, and can be executed, written to, and read by-all users), the disk sector the directory (or file) is located on, how many bytes the file contains (in hexadecimal), and the file name. If we had used lowercase letters to name this directory, we'd have to look at the attribute report to discover that it is a directory and not a file!

2.3 Let's play with the attr command some more:

OS9: chd personal <ENTER>
OS9: attr letters <ENTER>

The attr command looks on disk for the attribute report on the file "letters".

2.4 Now type:

OS9: chd letters <ENTER>
OS9: dir e <ENTER>

The "**dir e**" command will give you everything you want to know about the LETTERS directory's contents. Since LETTERS is empty, don't expect much.

```
 Directory of . 01:35:00

Owner Last modified Attributes Sector Bytecount Name
----- ------------- ---------- ------ ---------- ----------
```

2.5 So let's try using the attr command to reset (turn off) the directory attribute of PERSONAL:

OS9: chd ...<ENTER>
OS9: attr personal -d <ENTER>

Attr will now try to reset or turn off the directory attribute. If OS-9 actually did this, you could then use the del command to delete the file. However, since PERSONAL is not empty (it contains the LETTERS directory), 0S-9 will not let you reset the directory attribute.

2.6 So let's try deleting LETTERS:

OS9: chd personal <ENTER>
OS9: attr letters <ENTER>
OS9: del letters <ENTER>

This won't work either. Since LETTERS has its directory attribute set, you must either reset it or use the deldir command to delete it.

2.7 For right now, let's reset the directory attribute. Since LETTERS is an empty directory, OS-9 gladly resets it for us:

OS9: attr letters -d <ENTER>
OS9: del letters <ENTER>
OS9: chd .. <ENTER>

2.8 Now that we have successfully deleted LETTERS, the PERSONAL directory is empty. Let's delete this directory:

OS9: del personal <ENTER>

Did you get an error? You know what to do. Hint: see step 2.7.

2.9 Now let's use deldir — the easy way to do all this. First, re-create the directories we just deleted:

OS9: makdir PERSONAL <ENTER>
OS9: makdir personal/LETTERS <ENTER >

2.10 Delete the directory by typing:

OS9: deldir personal <ENTER>

You're on your own now!. You will be given choices to make. Use your judgement. The interactive nature of the deldir command may keep you from deleting valuable files and directories.

Step 3: Renaming

STEP 3: RENAMING FILES AND DIRECTORIES

3.1 Let's first examine the effect of lower and upper-case letters in naming and renaming files (the files in this case are directories). Make the following directories:

OS9: chd .....<ENTER>
(yes, that's five dots!)
OS9: makdir TestFile <ENTER>
OS9: makdir testfile/TestFile.sub <ENTER>
OS9: dir <ENTER>

Notice that OS-9 named your f ile exactly as you typed it. However, you named this directory in mixed-case (both lower-case and upper-case letters) which — by convention — is incorrect style for a directory name. You can't tell whether this is a directory or file by typing dir — you must use dir e then examine the attribute report. Have you memorized the report sequence yet? No? So you have to refer back to the manual.

Sure does take a while, doesn't it?
3.2 Let's rename TestFile to follow the OS-9 naming convention: **OS9: rename testfile TESTFILE <ENTER>**
TestFile can be typed "testfile" in the line above because 0S-9 can READ file names in either lower-case or upper-case or any mixture. However, since renaming TestFile to TESTFILE involves a WRITE to disk, 0S-9 is sensitive to the exact letters you type.

3.3 Take a look at what you did: **OS9: dir <ENTER>**
You can now easily tell that TESTFILE is a directory.

3.4 Now that TESTFILE looks like a directory, let's change TestFile.sub:
OS9: rename testfile/testfile.sub testfile/TESTFILE.SUB <ENTER>

Why did we receive an error? Doesn't it make sense to use the entire pathname for both the old f ile name and the new one? Perhaps it makes sense from a path name standpoint, but it does involve some unnecessary typing. 0S-9 tries hard to incorporate short-cuts, and the syntax of the rename command is no exception.

3.5 When you rename a file or directory, the second parameter is the local file name you desire. Try this:
OS9: rename testfile/testfile.sub TESTFILE.SUB <ENTER>

That should have worked fine. You may want to look at the directory just to see.

Step 4: More file attributes

STEP 4: ONE LAST
EXERCISE WITH FILE ATTRIBUTES

4.1 Type the following lines:

OS9: chd /d1 <ENTER>
OS9: attr cmds.s09 <ENTER>

4.2 The directory CMDS.S09 is an execution directory. It contains special utilities for use with "Mastering OS-9 " — in fact, use them anywhere. They are fun and extremely useful. Applause to Stephen Goldberg, who donated them to "Mastering 0S-9". But what's wrong with this picture? Why are there no e's in the attribute report? Because I reset them when preparing the disk.

4.3 As it stands, you cannot chx to CMDS.s09. Try it:

OS9: chx /d1/cmds.s09 <ENTER>

Use the error command to determine what error #214 means.

4.4 Since you are the super-user of your CoCo, you can set the execute attributes on this directory. We'll be using its commands heavily in the upcoming tutorials:

OS9: attr cmds.s09 e pe <ENTER>

If you were the super-user on a multi-user system, or if you ran an electronic bulletin board system (BBS) on your CoCo 3, resetting public execute attributes can be handy. A little security can go a long way!

4.5 Remove all disks from the drives and power down your system. Another tutorial has been finished!

GETTING STARTED WITH TUTORIAL 4

Tutorial 4 What we'll cover

We examine only three commands in this Tutorial. One of them (**edit**) invokes a powerful text editor that warrants most of our attention. We also formally introduce the **list** command mentioned extensively in previous reading. The third command, **tmode**, is perhaps most often used in conjunction with the list command.

You should have a list of commonly used phone numbers next to your computer for this and subsequent tutorials. These serve as good practice in entering data into the OS-9 text editor. They also constitute an ongoing project for these tutorials. When you're done with "Mastering OS-9", you'll have a handy phone directory at your fingertips.

If you're eager to type, include addresses and occupations for each person in your phone list. We can then show you a quick way to create a list of your doctors or perhaps baby-sitters. All it takes is one command line!

What editors are supposed to do

When you create text on your computer (letters, memos, lists, and so on) you insert text on your computer screen. If you make an error, you can change or delete that part of the text. You also need to move backward or forward in the file since flawed text may occur anywhere in your text file. To do all this, you'll need some kind of editor. Most editors:

1) Create text.
2) Move up or down one or more lines in the text.
3) Move forward and backward one or more characters.
4) Change text from <old text> to <new text> .
5) Delete lines and characters.

There are as many ways to implement these features as there are programmers. Below we discuss two general approaches.

Differing editors: screen oriented versus line oriented

Most editors allow you to move your cursor anywhere you wish on your screen with simple keystrokes, typically with your keyboard arrow keys. Move to any faulty text appearing on your screen, make your change, and you're done. This type of editor is the screen-oriented editor. While powerful and easy-to-use, these programs use up large amounts of memory.

The Macro Text Editor provided with OS-9 (from here on referred to as "edit") is a line-orinted editor. It is compact and powerful, in many ways surpassing in features many conventional text-editors.

Commands you will use -- **edit**

Being a line-oriented editor means you usually insert text, make changes, and view your text one line at a time. Short commands move edit's invisible cursor through text which may or may not be on the screen. Once edit's invisible cursor is near text that needs changing, short commands change or delete that text. Summarizing, OS-9's edit responds to short commands which

* create text
* move around your text one line at a time
* change the text in the given line to your new text
* delete a line or more of text

Edit therefore meets standard editing requirements. It also has many more powerful features beyond the scope of this book. To learn more pick up the Tandy OS-9 Level 2 manual where they present detailed lessons on edit's full capabilities

Entering **edit**

Typing "edit <filename>" at the "OS9:" prompt starts the OS-9 Macro Text Editor, or simply **edit**. At this point, a new prompt ("E:") replaces the familiar "OS9:" prompt on your screen. The "E:" prompt reminds you to type edit commands and not Shell commands.

One of your disk drives should be active, searching for a text file named <filename> in your current data directory. If it exists, edit loads the file and allows you to edit it. If there is no file named <filename>, edit creates an empty file in the current data directory with that name. Don't try to edit a command file. The machine codes create havoc on your screen and may hang up your computer.

If the file to edit is larger than edit normally handles (say, more than a page of single-spaced text), you can start Edit with a larger memory buffer. Type: **OS9: edit #24k <ENTER>** to set up Edit's buffer to 24 kilobytes (around 7 or 8 pages of single-spaced text).

Commands that move the edit pointer

Edit has two basic modes in creating a perfect text file: writing text and revising text. Writing text is done in the Insert Mode. Revising text is done in the Edit Mode. Whenever you start typing right at the "E:" prompt you are automatically in the Edit Mode, ready to revise text. Enter the Insert Mode by pressing the spacebar (denoted < SPACE> in this book) at the "E:" prompt, like so:

E: <SPACE> This is our first line of text. <ENTER>

Whenever you finish a line, press the <ENTER> key to get ready for the next line. An invisible marker called the "edit pointer" is advanced each time you enter a character or press the <ENTER> key.

Edit normally accepts up to 126 characters on a line. This is enough, say, to input a person's name, address, occupation, and telephone number. For our purposes let's keep lines short (less than eighty characters) to keep the screen from getting too cluttered.

Having finished entering your "rough draft", you may now be interested in maklng changes and deletions. These actions require edit commands.

The table below summarizes edit commands which move the invisible "edit pointer". These commands are logical and should be easy to remember.

edit commands that move the edit pointer

You must type:	In order to:
- (hyphen key)	move edit pointer back one line
+ (plus key)	move edit pointer forward one line
n (a number)	move edit pointer back "n" lines
+n (n=a number)	move edit pointer forward "n" lines
-* (hyphen-asterisk)	move edit pointer to the beqinning of your text
+* (plus-asterisk)	move edit pointer to the end of the text currently in your memory buffer

edit commands that change and delete text

If a line in your text file contains a small error, you can easily change the old text to new, correct text. Or, if the error is complex, you may want to delete the line and start all over again in the Insert Mode. Study this chart:

You must type:	In order to:
c/original text/new text/	change an occurrence of "original text" to "new text"
cn/original text/new text/	change the next "n" (number) of occurrences of "original text" to "new text"
c*/original text/new text/	change all occurrences of "original text" to "new text", starting from the position of the edit pointer.
D or d	delete the current line
L or l	list the next line in your file
L* or l*	list rest of file in the buffer

Using the asterisk

Notice the asterisk ("*") following the C command in the preceding command chart. C is obviously short for "change". What does the asterisk represent?

The asterisk can be loosely translated as "all". It can be used to mean "change all [occurrences of a string after the edit pointer]" as in the example above. It can be used with D to "delete all [file contents after the edit pointer]". It can be used with L to "list all [the file after the edit pointer]".

If you are at the top of a file, typing **E:d*** will "delete all" text. **E: l*** means "list all" the file from the present pointer position. To list the entire file, first place the edit pointer at the top of the file. Do this by typing: **E:-* <ENTER>**. Once the edit pointer is at the top of the text file, use the "list all" command: **E:l* <ENTER>.** We'll get first-hand experience with the asterisk in the tutorials, where you will find it a useful, easy way to enhance the normal function of edit commands.

Leaving edit

Once your session with Edit is complete, you will want to leave it and return to the "OS9:" prompt. To do this simply type **E:q <ENTER>** at the edit prompt. The "q" stands for "quit".

Listing files from the Shell Commands you will use -- **list**

We leave our discussion of Edit for now to introduce at last the list command. Type: **OS9: list filename <ENTER>** to list a text file named "filename" to your screen. If filename is not in the current data directory use a fuller path name so OS-9 can find the file. If OS-9 cannot find your file, it prints ERROR #216 (Path Name Not Found).

We briefly mentioned paths in the introductory reading. OS-9 sets up three paths for data flow: the standard input path, standard output path, and standard error path. Your screen constitutes the standard output path.

Listing files to the printer with redirection

One consequence of OS-9's Unified I/0 is that you are able to redirect the standard output path to devices other than your screen. You do this at the Shell's "OS9:" prompt with the output redirection modifier (>). For example, instead of viewing a listing on your screen by typing: **OS9: list filename** you can print the same listing on your printer by typing **OS9: list filename > /p**. This means "list the file named filename to the printer". Don't forget the slash preceding the device name.

Pausing a listing by changing output

Commands you will use -- **tmode**

Your screen ordinarily holds up to 24 lines of text. Listing a file longer than 24 lines causes the top part of the file to scroll off the screen. This is a nuisance when you need to read a file carefully, so you'll want to pause the display when the screen fills. Doing this takes a simple command line:

OS9: tmode .1 pause <ENTER>

This tmode command line offers a temporary way to change the way text is put out of your system. Although this output can go to many destination devices (for example, your screen, a printer, or an external terminal), most data is output to your screen, the standard output path. OS-9 assigns the standard output path the number one (1). This path is referenced by tmode when the period key (< . >) followed by the numeral one (< 1 >) is included after the command. Thus, the command line above pauses the display of data to the standard output path (your computer screen).

Turning pause off

Under some circumstances you do NOT want the screen display to pause when it is full. Many OS-9 programs stop activity when the screen fills. They will not continue unless you press a key (other than the <BREAK> key). Setting the screen to pause (OS9: tmode .1 pause <ENTER>) may force you to attend to your CoCo in case an OS-9 utility needs you to press a key. A good example: formatting a hard drive requires that screen -pause be off. Hard drive formats easily fill up a screen and simply won't continue when data fills a paused screen.

Let's move on to Tutorial 4. Don't forget to have your phone list handy!

Tutorial 4
Step 1: Inserting text

TUTORIAL 4
STEP 1: INSERTING TEXT WITH EDIT

1.1 Boot OS-9 with Custom Disk 1.

1.2 Create an eighty-column text screen (window) by typing:
OS9: window.t80s <ENTER>
Enter the screen or window by pressing the <CLEAR> key.

1.3 Now, looking at the eighty-column screen, type:
OS9: montype x <ENTER>
Replace the "x" with an "r" if you have a color RGB monitor (such as the Tandy CM-8 or Magnavox 8CM515). Replace with "m" if you have a monochrome monitor capable of displaying eighty-column text. If you are using a color composite monitor, you would replace "x" with "c". Most color composite monitors, however, will not adequately display 80 column text. If you are stuck with a color compoite for now, use montype m anyay and turn the color control until no colors are displayed. This turns your color monitor into a black and white monitor. Using montype m will turn the color burst signal off.

1.4 Find a disk label and a formatted, blank floppy disk. Write "Phone List and other files" on the label; place the label on the disk. Format the disk if necessary (I told you having a couple extra formatted disk on hand would come in handy later!).

1.5 Place the disk in drive /dl and close the drive door. Now change your current data directory to the root directory of that disk: **OS9: chd /dl <ENTER>**

Since you are using a blank, formatted disk, there is nothing in the root directory. If you wish, type dir at the "OS9:" prompt to check this.

1.6 Let's enter edit and create a new file:
OS9: edit Phone_LF <ENTER>

1.7 Before we start entering text, try to imagine the progress of the invisible edit pointer. It begins at the very top of the buffer. When you press "F", it moves just one space past that character. Then you press "i" and it moves one space further along in the buffer. The edit pointer continues to move along the first line until you press <ENTER >. Then it advances to the beginning of the next line. Type the following:
E: <SPACE> First line <ENTER>

1.8 The following steps continue text insertion. Remember to press the spacebar when you see <SPACE . This brings you into the Insert Mode.

E: <SPACE> Second line <ENTER>
E: <SPACE> We'll delete this line <ENTER>
E: <SPACE> Fourth line <ENTER>
E: <SPACE> Fifth and last line <ENTER>

Step 2: Using edit commands

STEP 2: USING EDIT COMMANDS

2.1 Type: **E: L <ENTER>**
The L command, typed at Edit's "E:" prompt, signals that edit should list the line following the edit pointer. Since the edit pointer is placed at the beginning of the (empty) sixth line, the L command shows nothing.

2.2 Now type: **E: L* <ENTER>**
Likewise, the L* command, meaning "list all " the text from the edit pointer to the end of the buffer, shows nothing.

2.3 First, we must move the edit pointer to some place closer to the beginning of the text. Type: **E: - <ENTER>**
The edit pointer moves back to the beginning of the previous line (line 5). Edit prints the line of text from the edit pointer to the end of the line. You will see "Fifth and last line" listed to your screen.

2.4 Now let's move the pointer to the beginning of the file:
E: -* <ENTER>
This command moves the edit pointer "all the way back". Now that the edit pointer is at the f irst character of your text file, edit displays the line of text from the edit pointer to the end of the line. This is, of course, the first line. You will see listed to your screen:

```
First line
```

2.5 Let's play with the List (L) comand some more. Type:
E: L<ENTER>. You will see:

```
First line
```

Now type: **E: L*<ENTER>**. You will see:

```
First line
Second line
We'll delete this line
Fourth line
Fifth and last line
```

2.6 Important: using the L command does not move the edit pointer. To prove that the edit pointer is still at the top of your text, type: **E: L<ENTER>**

Step 3: Moving the edit pointer

STEP 3: MOVING THE EDIT POINTER

3.1 Move the edit pointer to the beginning of the next line with:
E: + <ENTER>

3.2 Type: **E: +<ENTER>** again.
You should now see:

```
We'll delete this line
```

3.3 Now let's go back to the top of the text buffer:
E: -* <ENTER>

3.4 Hit the ENTER key twice. Notice that using the + command and hitting the <ENTER> key has the same effect.

Step 4: Deleting text

STEP 4: DELETING TEXT

4.1 Let's delete a line. If all has transpired as predicted, the edit pointer should be at the line reading "We'll delete this line".
Type: **E: d <ENTER>**
The command for deleting a line is **D** or **d**. Specifically, this command has deleted the line from the edit pointer (which is at thebeginning of the line) to the end of the line. Edit prints to your screen the line it just deleted.

4.2 So now we'll go back to the beginning of the text buffer and list our file:
E: - * < ENTER>
E: l * <ENTER>

You should now see:

```
First line
Second line
Fourth line
Fifth and last line
```

4.3 Let's reinsert the line we just deleted. Move the edit pointer to the beginning of "Fourth Line".
E: <ENTER>
E: <ENTER>

You should now see:

```
Fourth line
```

The edit pointer sits right before the "F" in "Fourth".

4.4 Type:

E: <SPACE> We'll zap this one again. <ENTER>

4.5 Move to the beginning of the buffer and list the file:

E: -* <ENTER>
E: L* <ENTER>

You should now see:

```
First line
Second line
We'll zap this one again.
Fourth line
Fifth and last line
```

4.6 Now move the pointer to the third line and check your work by displaying the lines before and after the former third line:

E: <ENTER>
E: <ENTER>
E: d <ENTER>
E :- <ENTER>
E: <ENTER>

4.7 Move the edit pointer to the beginning of the buffer then forward two lines:

E: -* <ENTER>
E: +2 <ENTER>

You should see:

```
Fourth line
```

4.8 Type in the following new line:

E: <SPACE> Line three <ENTER>

Where is the edit pointer now? To f ind out, immediately type:

E: L <ENTER>

As we noted at the beginning, our edit pointer was pushed forward with each keystroke; after we typed the new line, the edit pointer was pushed to the beginning of the fourth line.

4.9 Let's try that again, just to make sure you understand what's going on:

E: - <ENTER>
E: d <ENTER>
E: <SPACE> Line three again. <ENTER>
E: -* <ENTER>
E: L* <ENTER>

Step 5: Changing text

STEP 5: CHANGING TEXT

5.1 Let's change some text:

E: c*/line/line goes here/ <ENTER>

This command is straight forward. It simply means "change all occurrences of 'line' to 'line goes here' ". Edit is case-sensitive so it will not change line three, which should have the word "line" will a capitalized "L". Note that Edit reports which lines were changed.

5.2 Now we need to determine the location of the edit pointer:

E: L <ENTER>

Edit moves only as far into the text as the last occurrence of the target text ("line").

5.3 Let's try changing some text again:

E: -* <ENTER> (go to beginning of text)
E: L* <ENTER> (list all of file)
E: c*/again./is what goes here/ <ENTER>

(change all occcurences of 'again.' to 'is what goes here')

E: L <ENTER> (list the line at the edit pointer)
E: -* <ENTER> (go to beginning of text)
E: L* <ENTER> (list all of file)

Step 6: Building a phone list

STEP 6: BUILDING A PHONE LIST

6.1 Now that you have experience inserting, deleting, changing , and listing text, let's start from scratch with our phone list. The first step is to clear the buffer of all text:

E: d* <ENTER>

Once again, the delete command prints to your screen all the text it has deleted.

6.2 Now begin entering your list of names and numbers in the Insert Mode. Recall that the Insert Mode is entered by pressing the spacebar at the "E:" prompt. So, press the space-bar, enter a line of data, then press <ENTER>. The backslash symbol ("\") is generated by pressing the <CTRL> key and the slash key ("/") at the same time. Enter all names and numbers in the following format, each list on its own line, like this:

Lastname, Firstname\occupation or note\phone # with area code
Swygert, Francis G.\Publisher, 68' micros\912-328-7859
FARNA Systems\CoCo Vendor\912-328-7859

For now, put in at least 25 names and numbers. You need that many lines in the list to properly demonstrate the tmode command in Step 7. If you don't know enough numbers , pick up a phone book and just put any in. You can delete them later.

STEP 7: USING THE LIST TMODE COMMANDS

7.1 Quit Edit with the following command:

E: q <ENTER>

7.2 After the file is saved, the Shell prompt reappears.

7.3 Type the following:

OS9: tmode .1 pause <ENTER>
OS9: list Phone_LF <ENTER>

Not all of your phone list will appear. Press any key (but the <BREAK> key!) to continue the listing.

7.4 Now let's turn pause off and list the file:

OS9: tmode .1 -pause <ENTER>
OS9: list Phone_LF <ENTER>

This time, the screen did not pause when it was full. How do you pause the screen output when tmode has not been set to pause?
Use the <CTRL> - <W> key combination during the listing (press and hold CTRL then press W). This pauses scrolling text until the next keypress. To help remember this combination, think of "W " as being short for "wait".

7.5 Let's try using the <CTRL> - <W> combination. List the phone list again, this time using <CTRL> - <W> to keep the list from scrolling past the top of the screen:

OS9: list phone_lf <ENTER>

As the text scrolls, type

<CTRL> - <W>

Press any key to continue the listing. Pressing BREAK will abort the listing. This is exactly like using "<SHIFT> - <@>" to halt a BASIC program listing under CoCo BASIC.

7.6 Remove all disks from drives when finished and power down. But before you do — when is the last time you backed up your data disks? If you accidentally format a disk or delete a file while we are going through the tutorials, you'll have to go back and start all over! You might want to make sure you have an extra copy (backup) of the Phone List disk just in case. Every time you add names to the list, back it up again. If you somehow loose you data file, you'll only have to retype what was entered between backups, not the entire list.

Tutorial 5: What we'll cover

GETTING STARTED WITH TUTORIAL 5

Tutorial 5 finishes the first "week" of "Mastering OS-9". This time, we explore procedure files. One such procedure file, named "startup", comes with the OS-9 system disk. Using edit, we'll change the startup file to meet the needs of those who own printers and disk drives capable of quick access. We learn about the xmode command which is like a big brother to the tmode command introduced in Tutorial 4.

Other commands and concepts in this Tutorial are familiar territory. In addition to edit, we again use the copy, del, and rename commands. If you are rusty on these, take a moment to review them. At the end of Tutorial 5, we ask that you add to the Phone_LF text file created in Tutorial 4. Keep your personal list of phone numbers handy.

Procedure files

Now that you can use edit, you can create and edit a **procedure file**. Knowing what to include in a procedure file depends on an understanding of the different ways the Shell accepts commands.

The Shell normally processes your commands one at a time as you enter them from your keyboard; thus, the keyboard is the standard input path, as we have discussed before. The Shell can receive commands from a source other than your keyboard, too. In these cases you are witnessing an example of input path redirection. The Shell's execution of procedure files from the command line is a case of input redirection.

The most common form of input redirection you will encounter is the procedure file. The Shell begins processing command lines in a procedure file almost as soon as you type the procedure's pathname and press <ENTER> (the only business the Shell takes care of first is making sure that the pathname is not a command but a procedure file). After the Shell determines that you want it to process a procedure file, it reads the file one line at a time. At this point, the Shell treats the text in the procedure file exactly as it treats commands you type in from the keyboard.

One way to work is to keep your most commonly-used procedure files in the system disk's root directory. After boot-up simply type the name of the procedure file to execute it. What could be more convenient?

Command lines in procedure files

Procedure files can contain any valid command line. This may include commands and other utilities. It may even include other procedure files.

One work strategy is to write a handful of short procedure files each of which does one simple job. These become tools used to tackle larger jobs. Do this by stringing together the pathnames of appropriate procedure files into a single "master" procedure file. Execute the master file and the job is done.

A customized Startup file is a good example of this technique. Your boot disk's root directory contains a few procedure files which create windows of various colors and capabilitles. By including the pathnames of some of these procedure files in your startup file, you can automatically create a windowing environment. Tutorial 5 demonstrates how to include the procedure files "window.t80s" and "make_80" in your Startup procedure file. These two files create attractive eighty-column screens.

The startup file

OS-9 takes care of a lot of business before you see the first "OS9:"prompt. One of its tasks is to search in the root directory of drive /d0 (drive /h0 for hard drive users) for the startup file. When OS-9 finds it, startup is executed line by line.

Startup — a procedure file — can he as simple or as complex as you wish. On the Level 2 distribution disk, the startup file reads:

```
* Echo welcome message
echo  * Welcome to OS-9 LEVEL 2 *
echo  * on the Color Computer 3 *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
```

This file, executed one line at a time, has two simple but important purposes:

* It makes the Shell available to you at all times by linking it to the OS-9 system inside the CoCo.
* It prompts you for the time and date as part of its record- keeping.

Translating startup line-by-line

The Shell's first job when encountering startup is to translate it line-by-line. In Tutorial 5 we simulate startup's action in "slow motion" by typing each line ourselves at the "OS9:" prompt. To prepare you, let's preview that process.

Before we start, though, you should know that the Shell will not process any text which begins with an asterisk in the first column. Any text following a first-column asterisk serves as a comment. This text reminds the user what various parts of the file are designed to do.

line 1: *Echo welcome message
Since the first character in this line is an asterisk, the Shell skips this line and goes to the next. This line is here to tell us what the next lines will do.

line 2: echo * Welcome to OS-9 LEVEL 2 *
The OS-9 echo command simply prints to the screen all the text which follows it. This command line therefore prints "* Welcome to OS-9 LEVEL 2 *" to your screen.

The echo command cannot directly print all characters, though. Some characters are reserved for the Shell. For example, the following characters cause problems: **< > ! &**

These are Shell modifiers which redirect input and output and set up processes to be run concurrently. If they were included in a message typed after echo, the Shell would try to execute them and would not echo them to your screen. If you want these characters to appear in a message, follow the example which "window.t80s" provides:

```
echo Creating 80 column text window
*
* Create a 80 column text window using descriptor /w7
* using White letters on a blue background
* NOTE: Wcreate is commented out since
* the defaults for /w7 are used.
* wcreate /w4 -s=2 0 0 80 24 0 1 1
iniz /w7
echo Window /w7 >/w7
shell i=/w7&
* Print Message to user
echo "Press <clear> to select window screen"
```

Note especially the last line. The symbols < and > appear, and yet they do not cause problems. Why? The secret is in the quotation marks. They protect the enclosed text from interpretation by the Shell.

line 3: echo * on the Color Computer 3 *
The echo command here is used to finish the welcome message.

line 4: * Lock shell and std utils into memory
Once again, the asterisk marks this as a comment line. This line informs us that the next line will lock the Shell into memory for permanent use by OS-9. Since the Shell file contains not only the OS-9 Shell but many often-used commands which are merged into it, these standard commands are also permanently available. These often-used commands are what is meant here by "std utils" (standard utilities).

line 5: link shell
The link command increases the Shell's link count by one. This means that you would now have to work VERY HARD to get rid of the Shell in memory. Having the Shell permanently in memory is what you want, though, so linking it is a good idea.

line 6: * Start system time from keyboard
Here is the comment line belonging to the next line.

line 7: setime < /1
The setime command asks you for the current date and time. This time is entered in "military" or "European" format, with a 24-hour clock — but you know this by now, having entered this data several times..

Recall from the discussion of tmode that the standard input path is given the number one (1). The standard input comes from your keyboard. Thus the Shell translates " < /1" as "redirecting input from the standard input device".

line 8: date t
The date command returns the current date to the screen. If you add the "t" parameter it also returns the current system time. By the time the Shell gets to this line, you have entered the current time and date; now it displays it so you can make sure you entered the right information. As you know, date prints the date in "English", not in military format.

Possible enhancements

We created an eighty-column text screen in Tutorial 4 using "window.t80s" at the "OS9:" prompt. That procedure file may be included in your startup file. If it is added to startup, when the boot process is over you will have a new eighty-column text screen available with a press of the < CLEAR > key. This is in addition to the familiar 32-column green VDG screen (VDG actually stands for Video Display Generator, a term borrowed from previous Color Computer hardware). Other features can be added to your system by adding commands to the startup file, including:

l) Faster disk access
2) Faster printing speed
3) Eighty-column screen on boot-up
4) Additional screens (windows)
5) Personalized startup message
6) Automatic loading of your favorite utilities
7) Automatic execution of an application
8) Almost limitless additional features!

A specific example

Being able to automatically load and execute an application on bootup allows you to create a separate boot disk for each given job; this boot disk may AUTOMATICALLY load your favorite utilities into memory (for fast access) and then start the application best suited for your task in a screen of your own design.

For example, if you do a great deal of word processing, you may have purchased utilities — small programs — which:

* count the number of words in a file
* count the number of lines in a file
* find the average length of words
* find the longest word
* find the longest sentence
* find the longest paragraph
* ...and so on.

These utilities may be used to help your writing style respond to the reading level of, say, younger audiences by pointing out longer words, sentences, and paragraphs. You can load these automatically by adding command lines to the Startup file to do the job (you might first want to merge them for reasons covered in Tutorial 6). After the Startup file has loaded these utilities it might automatically execute your word processor in an eighty-column window set up just for it. All this with no additional typing from you!

Before editing the startup file, sketch out a plan of attack.:
l) Make sure we have a "backup" copy of the original startup file. Do this by copying the original to a new file named, say, startup.old. Extensive edits are then possible to the original startup file. If your editing session goes foul, you can always start from scratch with startup.old.
2) Edit the file. Use either edit or your own OS-9 based word processor.
3) Reboot OS-9 to test the new startup file. In some cases, rebooting is unnecessary; just type "startup" at the "OS9" prompt.
4) Didn't work correctly? Make any necessary changes and additions.
5) Repeat steps (3) and (4) until startup is perfected.
6) You're done.

(Those who are familiar with MS-DOS machines will notice that startup is exactly the same as autoexec.bat.)

How to optimize your printer

There is no use in your Color Computer sending text to your printer as fast as it can. Your computer is just too fast. The printer can become over-run with data inside of a second or two. The printer must have some way of telling your computer to stop sending data until it is ready again. This is done through handshaking — an electronic version of the human custom. When the printer is ready to receive more text from the computer, it sends out a handshake signal. It sends out another handshake signal when it is full of data and can accept no more.

Commands you will use -- **xmode**

Handshaking and transfer of text requires that the computer and your printer be communicating at the same baud rate. Baud (bits per second) refers to the speed at which data is transferred. While OS-9 defaults to a baud rate of 600, your printer can probably accept data faster than that. Most modern printers accept data up to16 times faster (9600 baud). Even Tandy's old DMP-105 can accept data at 2400 baud! Once you determine how fast your printer can accept data, use the **xmode** command to change the speed at which OS-9 sends data to the printer.

Note that most serial-to-parallel convertors can also be set to 9600 baud. Some have an adjustable setting. My Epson LX-800 printer (a 180 cps nine pin printer) is connected to my CoCo through an adjustable rate Botek (OLD!) serial-to-parallel convertor. I experimented with the settings under Disk Basic and found that sending it data over 2400 baud netted no

improvements in print speed. Therefore, I have my personal system set at 2400 baud. You can easily change the baud rate while OS-9 is running to determine your printer's fastest setting. Sometimes, setting it to fast will actually cause a reduction in print speed. If you have a hardware parallel port and not a serial printer or convertor, changing the baud rate will not affect print speed.

The **xmode** *command in general*

The **xmode** command changes values in the device descriptor tables in your CoCo's memory making xmode's effect powerful. After the baud rate for your printer has been set with the xmode command, every path opened to the printer sends data at the new rate.

To use xmode, you provide two pieces of information. The first is the name of the device or module you wish to affect. The second is the new value for its device descriptor table.

Example: Xmode provides detailed control over many device parameters. Suppose you need to print a report in all uppercase letters. Go ahead and edit on your screen in mixed case, but set your printer descriptor to print all uppercase with the following command line: **OS9: xmode /p upc <ENTER>**
Now, even though your screen shows "Quarterly payments less in-kind contribution", your printer will print "QUARTERLY PAYMENTS LESS IN-KIND CONTRIBUTION".

If you forget which parameters you can change in the printer's device descriptor, receive a report by typing:

OS9: xmode /p <ENTER>

You receive the following report:

```
-upc -bsb -bsl -echo -ff null=0 -pause pag=66 bsp=08
del=18 eor=0D eof=00 reprint=04 dup=01 psc=17 abort=00
quit=00 bse=5F bell=07 type=00 baud=06 xon=00 xoff=00
```

Notice some common words among the codes above. Other codes are suggestive of what they represent. For example, if you guessed "bsp" stands for backspace, you were right. You can find more complete explanations for each of these codes in the OS-9 Level 2 manual from Tandy.

The xmode command and you

You are spared having to learn the intricacies of xmode because it is typically used in only a few instances:
l) When you want to change your printer baud rate.
2)When your printer requires a line-feed with each carriage return.

3) When you want to create an extra VDG window.
The command line which changes the printer baud rate is:

OS9: xmode /p baud = nn <ENTER>

where you replace "nn" with the baud code corresponding to the highest baud rate your printer accepts. The following chart summarizes codes for each common baud rate:

Printer Baud Codes

Baud Rate	Code
600	02
1200	03
2400	04
4800	05
9600	06

In the following tutorial (Tutorial 5) we assume that your printer handles 2400 baud. We therefore edit our tartup file to contain the command:

OS9: xmode /p baud=04 <ENTER>

Let's start

Now that you know about procedure files in general and the Startup file in particular, we can move into Tutorial 5. Be prepared to edit startup and experiment. You should also have your phone list handy so we can add to your previous work.

NOTE: In Tutorial 5 we have you call "BoostStep" from the startup file in order to speed up your drive's stepping rate. Ordinarily, the drives step at 30ms (milliseconds) track-to-track, which can sound loud and unpleasant. BoostStep pushes the step rate down to 6ms, which most drives can easily handle.

If you have older drives which produce I/O errors when operating at 6ms, you will need to reboot your computer using a disk with the original startup. You can then edit the modified startup by replacing BoostStep with "BoostStep.20" or "BoostStep.10". These change the step rate to 20ms and 10ms, respectively. If 6ms fails, try 10ms. Even the oldest full height Tandy drives will handle 20ms; almost any newer drive should handle at least 10ms. All the "BoostStep" files are on the "Mastering OS-9" disk in the HANDY_PROCS directory.

Tutorial 5
Step 1: Copying startup

TUTORIAL 5
STEP 1: CREATING A
SECOND COPY OF THE STARTUP FILE

1.0 Boot OS-9 with Custom Disk Backup 1. Make sure it is not write-protected by removing the write-protect tab from the disk if one is present.

1.1 we now make an 80 column text screen:

OS9: window.t80s <ENTER>

Switch to the new screen with the <CLEAR> key.

1.2 Get a listing of the root directory of the disk in drive /d0:

OS9: dir <ENTER>

Notice the startup file

.1.3 Now let's get a listing of the current execution directory:

OS9: dir x <ENTER>

1.4 Which modules are loaded in memory?:

OS9: mdir <ENTER>

To remind you of your OS-9 "whereabouts", you just retrieved a directory from the current data directory with the dir command (in this case, the current data directory is your root directory).
Then you used dir x to see the current execution directory with all its commands Last, you used mdir to find the list of modules and commands in memory. Notice the edit command is not in memory.

1.5 To make a little more room on your Custom Disk l Backup, type the following lines:

OS9: del -x build cmp dcheck deiniz iniz
OS9: del -x date dir dislay echo
OS9: del -x tuneport mdir mfree

1.6 Since you'll be doing a lot of editing, go ahead and load edit. Also load xmode.

OS9: load edit xmode <ENTER>

1.7 We now need to make a spare copy of startup just in case we run into problems later:

OS9: copy startup startup.old <ENTER>
OS9: dir <ENTER>

There is your spare copy of the startup file.

1.8 Now type: **OS9: list startup <ENTER>**
You will see:

```
* Echo welcome message
echo  * Welcome to OS-9 LEVEL 2 *
echo  * on the Color Computer 3 *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
```

Step 2: Changin startup message and printer baud rate

STEP 2: CUSTOMIZING STARTUP MESSAGE AND BAUD RATE

2.1 In order to edit the file, we need to start edit and move the pointer to the beginning of the file:

OS9: edit Startup <ENTER>
E: - * <ENTER>

You will see:

```
* Echo welcome message
```

2.2 List the entire file:

E: l* <ENTER>

You will see:

```
* Echo welcome message
echo  * Welcome to OS-9 LEVEL 2 *
echo  * on the Color Computer 3 *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
```

2.3 We now explore how to change the welcome message:

E: + 3 <ENTER>

You will see
:

```
* Lock shell and std utils into memory
```

The edit pointer is now at the beginning of this line. We want to insert another message to be echoed before this line is executed by the Shell. So press the space bar (for Insert Mode), type echo, and then enter a message. For example, type:

E: <SPACE> echo * which can be really fun * <ENTER>
E: <SPACE> echo * (depite what they say) * <ENTER>
E:-* <ENTER>
E:l* <ENTER>

Check your work. Looks good! Now go back and change these lines (or even the original two lines) to whatever you want. After all, it IS your computer!

2.4 Now let's insert a few housekeeping lines after the "date t" command line. Since that line is almost all the way at the end of the startup file, advance to the end with

E: +* <ENTER>

which means "go all the way to the end of the buffer". To make sure we insert our command line immediately after date t, back up line by line with the "-" until you see: **E: date t**
Then type: **E: <ENTER>**

2.5 Now type:

E: <SPACE> montype r <ENTER>

if you have an RGB monitor. Replace r with m if you have a monochrome monitor. Insert this line next:

E: <SPACE> xmode /p baud = 04 <ENTER>

2.6 Oops! We forgot to add a comment line! We also forgot to add a line to echo to the screen exactly what the Startup file is doing. This is a courtesy to the user. Think about it: one day, someone other than you will start your OS-9 computer and wonder why the screen is empty. Without echoes to the user, your startup file could do ten minutes of "computer housekeeping" hidden behind a blank screen.

2.7 Type the following to move the edit pointer back one line:

E :- <ENTER>

You will see:

```
E:
 xmode /p baud = 04
```

Just where we want it to be!

2.8 Now we'll insert the comment lines:

E: <SPACE> * xmode changes the printer <ENTER>
E: <SPACE> * rate to 2400 baud <ENTER>
E: <SPACE> echo Boosting baud to 2400 <ENTER>
E :-* <ENTER>
E: l* <ENTER>

Once again, we check our work with the "l*" command. You

should see:

```
* Echo welcome message
echo  * Welcome to OS-9 LEVEL 2 *
echo  * on the Color Computer 3 *
echo * which can really be fun *
echo * (despite waht they say) *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
montype r
* xmode changes the printer
* rate to 2400 baud
echo Boosting baud to 2400
xmode /p baud=04
```

2.9 Now we can quit the editor:

E: quit <ENTER>

2.10 Okay, let's check and see if this thing really works! First, we'll see what xmode is currently set to:

OS9: xmode /p <ENTER>

Notice the baud rate reads 02.

2.11 Now let's run startup:

OS9: startup <ENTER>

Enter the correct time and date.

2.12 We should have a clearer screen from the "montype" command. Let's check the baud rate:

OS9: xmode /p <ENTER>

Notice that the baud rate now reads 04, which translates to 2400 baud.

Step 3: Testt startup, more with **xmode**

STEP 3: TESTING STARTUP AND PLAYING WITH XMODE

3.1 Set your printer or serial-to-parallel convertor up to 2400 baud if you haven't done so already (of course, if your printer handles a higher baud rate, you will have to change the Startup file as indicated in the introductory text). Turn the power on and get it ready to print. Consult your printer manual if necessary. Now let's print a copy of the startup file:

OS9: list startup > /p <ENTER>

3.2 Let's see what else we can do with xmode:

OS9: xmode /p upc <ENTER>

OS9: list startup > /p <ENTER>

Compare the printouts. See how easy it is to change the printer device descriptor? Change the descriptor back by typing:
OS9: xmode /p -upc <ENTER>

3.3 Turn the speaker up on your monitor if you have one. Now type: **OS9: display 7 <ENTER>**
You should hear a beep, called the "bell". The numeral 7 is the computer (ASCII) code commonly used to sound a device's beeper or bell.

3.4 Now try this: **OS9: display 7 > /p <ENTER>**
If your printer has a bell, it will ring or beep (if not, nothing should happen). Here we redirect the output of the display command with the " > " symbol. In this case, we redirect the output to the printer /p.

3.5 Check the xmode parameters:
OS9: xmode /p <ENTER> You hould see:

```
upc -bsb -bsl -echo -lf null=0 -pause pag=66 bsp=08
del=18 eor=0D eof=00 reprint=04 dup=01 psc=17 abort=00
quit=00 bse=5F bell=07 type=00 baud=06 xon=00 xoff=00
```

Notice the bell parameter is set for 7. Refer to Tandy's OS-9 manual for explanations of the other xmode prameters.

Step 4: Adding windows to startup

STEP 4: ADDING WINDOWS IN THE STARTUP FILE

4.1 Let's edit our startup file once again to have it create an 80-column screen. We'll also convert the 32-column green VDG screen into a black-and-white 80-column screen. We do this by naming two other procedure files in the startup file. This demonstrates that procedure files can call other procedure files.

4.2 We'll start by calling edit and loading startup into its buffer. Then we'll move the edit pointer to the top of the file and list it:
OS9: edit startup <ENTER>
E: -* <ENTER>
E: l* <ENTER>

4.3 Where do we want to place our new command lines? How about at the very end of the existing Startup file. Don't forget we have to comment our work and echo to the user what the system is doing:
E: +* <ENTER>

E: - <ENTER>
E: - <ENTER>
You should see:

```
E:
 xmode /p baud = 04
```

Place the edit pointer after this line by pressing <ENTER>

4.4 Now we can insert our new text:
E: <SPACE> * Change current window <ENTER>
E: <SPACE> * to 80-column text window <ENTER>
E: <SPACE> echo Changing terminal <ENTER>
E: <SPACE> make_80 <ENTER>
E: <SPACE> * create blue 80-col screen <ENTER>
E: <SPACE> echo Setting up New Window <ENTER>
E: <SPACE> echo and Shell <ENTER>
E: <SPACE> window.t80s <ENTER>
E: <SPACE> merge sys/stdfonts > /w <ENTER>

Notice in particular the last line. This command line makes available the standard text fonts for graphics screens. You cannot merge sys/stdfonts or any other fonts to a VDG (32-column green screen) device. I also advise you to merge fonts into your system BEFORE initializing a graphics window with wcreate, iniz, or display.

The reason for these two rules: First, VDG screens emulate the older CoCo 1 and 2 video hardware, which knew nothing about windows much less graphics windows. Only when windows are opened does OS-9 worry about which fonts are in which buffers. Secondly, if you initialize a graphic window WITHOUT merging any fonts into memory, the screen will only display dots where characters should be.

4.5 Now let's check our work:
E: -$ <ENTER>
E: l * <ENTER>

You should see:

```
* Echo welcome message
echo  * Welcome to OS-9 LEVEL 2 *
echo  * on the Color Computer 3 *
echo * which can really be fun *
echo * (despite waht they say) *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
montype r
* xmode changes the printer
* rate to 2400 baud
echo Boosting baud to 2400
xmode /p baud=04
* Change current window <ENTER>
* to 80-column text window <ENTER>
echo Changing terminal <ENTER>
make_80 <ENTER>
* create blue 80-col screen <ENTER>
echo Setting up New Window <ENTER>
echo and Shell <ENTER>
window.t80s <ENTER>
merge sys/stdfonts > /w <ENTER>
```

4.6 That completes our editing session for now, so we can leave the editor:

E:q <ENTER>

STEP 5: ALTERING KEY REPEAT AND DISK DRIVE STEP RATE

5.0 If you have drives which can handle a 6ms step rate (and all modern disk drives can), you may wish to include a procedure in
your Startup file to boost your step rate to that speed. Such a file, BoostStep, is included on your "Mastering OS-9" diskette in the HANDY_PROCS directory. There is a better, more permanent method to change the drive step rate which we will show you later in one of the appendix articles.

Also included is a short procedure file, MudKeys, to delay a little longer before repeating the current key press. Slowing key repeat can avoid accidental doubling or tripling of characters on your screen.

5.2 Place the BACKUP of the "Mastering OS-9" disk in drive /dl and type:

OS9: chd /dl/handy_procs <ENTER>
OS9: copy MudKeys /d0/sys/MudKeys <ENTER>
OS9: copy BoostStep /d0/sys/BoostStep <ENTER>

This copies the files we need from the Mastering OS-9 backup disk to Custom System Disk1 Backup, which should still be in drive /d0.

5.3 Now we need to change our data directory back to drive /d0 and edit Startup again:

OS9: chd /d0; edit startup <ENTER>
E: +* <ENTER>
E: - <ENTER>

Continue the above step (E: -) until the edit pointer is at the beginning of the following line of text:

xmode /p baud=04

5.4 We want the pointer to be right after the above line, so we need to press <ENTER> one more time. We can then insert the following text:

E: <SPACE> sys/BoostStep <ENTER>
E: <SPACE> sys/MudKeys <ENTER>

5.5 Check your work by listing the startup file. You should be familiar with how to do this by now!

5.6 If all is well, quit the editor: **E: quit <ENTER>**
If you made a mistake, go back and fix it!

5.7 Press reset to test the new startup. When reset is pressed once, OS-9 tries to boot from the disk in drive /d0. If reset is pressed twice, OS-9 aborts and control is returned to Disk Basic.

5.8 If all works well, type: **OS9: del startup.old <ENTER>**
A copy of the original startup file is still on your original System Master and your System Master Backup.

Step 6: Adding to phone list

STEP 6: ADDING TO THE PHONE LIST

6.1 Place the disk marked “Phone List” in drive /d1 and change the data directory to that disk:

OS9: chd /d1 <ENTER>

6.2 Enter the editor to edit the file Phone_LF:

OS9: edit Phone_LF <ENTER>

6.3 Move to the end of the list with the appropriate Edit commands. Add at least six numbers to the list. We need a long list later in the tutorials for demonstartion purposes. You can use the practice with Edit also!

6.4 When finished adding names, backup your data disk. When the backup process finishes, remove all disks and power down.

I don't know about you, but all this work sure has made me want a break! And that's exactly what you're getting... sort of. The next couple of sections are essays on just what you can do with OS-9, some handy items to keep in mind, and its future.

If you are considering skipping the next sections, please don't! Some very good information will be presented that I'm sure you'll want to know. If I weren't, it wouldn't be here!

So relax and enjoy some light reading for a couple days! We'll get right back to work afterwards.

CoCo OS-9 Applications

Application Programs for OS-9 on the Color Computer

Of the tens of thousands of application programs available for all computers, most fall into five categories. I call these "The Big Five":

: 1) word processors
2) spreadsheets
3) computer languages
4) utilities
5) telecommunications sof tware

OS-9 on the Color Computer offers strong contenders in each of these fields. Before you choose a particular brand read through this essay where these questions are answered:

How can these applications help me?
How do they work?
Who offers these applications?

Products are mentioned only for purposes of illustration. No endorsements are intended nor implied. Before you buy, ask around. Everybody's needs are different — that's why so many programs are written!

Word Processors

Word -processors usually come in two parts under OS-9 to help preserve the OS-9 "Unified I/O" philosophy and to maximize use of memory. The first part is the text editor. An editor usually allows your cursor to move freely around your computer screen so you can add, delete, and move text. You can also add symbols in the text which tell the formatter (commonly offered as a companion to the editor) exactly how to print the text on your partlcular printer. The formatter translates special symbols within your text into codes the printer understands. Keeping the main text free of printer-specific codes allows the text to be easily manipulated by other programs. Often you can buy the editor separately from the formatter.

Most OS-9 word processors (DynaStar and Ved, for example) are modeled closely after the infamous WordStar (tm), made popular first on C/PM machines and then IBM compatibles. They are, in the lingo of computer salesmen, "similar in functionality". A person who knows the WordStar keyboard commands (held over from the earliest to the latest versions for the benefit of upgraders) has an easy transition when moving to these word processors.

Ved and its companion text formatter, Vprint, are still available from Bob van der Poel. Color Systems markets a specialized

"point-n-click" shell for use with your text editor, formatter, and related utilities (such as a spell-checker). While this shell isn't a word processor itself, it does serve to integrate the editor, formatter, spell-checker, and other utilities of your choice into a very easy to use package.

You can also choose from several programs to transfer text files from your OS-9 CoCo to a PC compatible running MS-DOS. I know many OS-9 users who retain their OS-9 home system to complete office work created on a PC.

Spell-checkers, especially useful under OS-9 since you can easily check the spelling on a document in the background while continuing your work. The most famous is Dale Puckett's DynaSpell. You can add to the exising DynaSpell dictionary by purchasing spell checking dictionaries from the OS-9 Users Group. At last look, the User's Group library also contained a spell-checker, handy if you can't find a copy of DynaSpell.

Spreadsheets

Spreadsheets are advanced electronic calculators that finally bring easy financial planning to your desktop. They divide your screen into rows and columns. The intersection of, say Row A and Column 1, is a cell (named "A1" in this case). Numbers and mathematical formulae are entered into cells to create a custom calculation system. Change one number and any other number derived from it will change automatically, allowing you to test a variety of values to arrive at an optimal answer. Most OS-9 spreadsheets offer several ways to transfer their data to other OS-9 programs (such as word processors) frequently including only results and ommitting formulas.

By far the most popular OS-9 spreadsheet is DynaCalc. It has lightning-fast recalculation, large spreadsheet size, windowing features, and on-line help. It is an extended remake of VisiCalc (tm) and feels much like Lotus 1-2-3 (tm).

DynaCalc is still available through Radio Shack Consumer Mail Centers. Ask to see the CMC catalog at any RS and they can order it (and many other Tandy OS-9 titles) for you.

Databases

Databases usually allow you to create screen forms for data entry. This data can be of most any type (names, numbers, dates, and so on). You can also create reports summarizing data. Printing formats such as mailing labels and envelopes are commonly supported.

Data is generally assembled in a file containing individual records. A record for your personal medical history is included in a master medical file, for example. The place on the form where your last name is entered is a f ield. Thus, data about you goes in fields. All of these fields together make up your record, and all related records are a file.

Most databases allow you to create screen forms to keep the file up-to-date. You can often create report forms, too, for printing results.

More powerful databases create relational indexes which allow quicker access to your data. An index for one database (say, Widgets) can be linked to another file's index (say, Customer Info), allowing you to access information in several database files at once. In this example, you could easily keep track of how many customers you have, how many widgets are in stock, and which customers bought widgets. Linking relational indices makes your data "intellient".

In conjunction with a query language (similar to BASIC or Pascal), you can often find and update data in surprisingly useful ways. The database I use, for example, includes an interactive mode in which the "query" command line:

list all for occupation ct "Dentist" <ENTER>

gives me a list of all dentists in the database.

To give a more complex example, these languages can be used to create a short program which will search for a given field (say, Earned Income or Donations 1994), perform a calculation, print a special report, and then delete the record. The query language is often so close to English that its learning curve is even softer than BASIC.

One database package, Data Master from Computerware, while reportedly not fully relational, is the only database program I know which was written expressly for CoCo 3 OS-9 Level 2 with Windows.

Two popular and capable relational database programs are the Clearbrook Software Group's Information Management System (IMS) and Sculptor . Each is a strong package and highly competitive in performance.

Languages

Languages, including the powerful Basic09 language that comes with OS-9 Level 2, usually produce text files which you can edit and perfect. These "source files" are usually changed by a compiler into the codes your computer needs. Basic09 implements a sort of "half-compiled" intermediate code called i-code which runs incredibly fast for a Basic language. In addition to Basic09, OS-9 supports excellent C and Pascal compilers from Microware.

The biggest seller (other than Basic09) is surely the C compiler. The last version came with the OS-9 Level 2 Development Package and includes handy features for professional and beginning C programmers alike.

For a more offbeat and fascinating language, take a look at FORTH09 from D.P. Johnson. Johnson worked hard to follow OS-9's "rules" and has produced a ROMable, reentrant, and relocatable FORTH code generator. Code produced by this package is small and very fast. The main D.P. Johson FORTH09 module has now been released to the public domain and is available for downloading on Delphi and possibly other services supporting the CoCo. D.P. Johnson continues to sell specialized modules for FORTH09.

The OS-9 assembler is one of the most advanced assemblers on any microcomputer. It helps protect from creating OS-9 incompatible code through system call "equates". These equates can also speed assembly programming as they offer short-cuts to accessing the computer. The assembler will be found in the OS-9 Level II Development Package or with the OS-9 Level I package. The Level I assembler is the same as that found in the Level II Dev Pack. That is the primary reason many OS-9ers pick up Level I (if they don't already have it). The Level II Developers Package costs much more than Level I and is harder to find.

OS-9 provides one of the leading development environments on any computer system. Level 2 with windows can speed program development substantially. The resulting programs offer high portability. BasicO9 programs written on your CoCo can almost always be run on any other OS-9 computer, including the MM/1 and an OS9/68000 (OSK for short) based Atari ST. Most Microware C programs can also be compiled and run without changes on large UNIX based mainframes. This testifies to the programming power in your OS-9 Color Computer.

Utilities

Utilities, like the Goldberg utilities offered with this book, frequently accept your data files as input and then produce another file as output.

For example, suppose you want to sort the contents of your phone list database we have been working on. You can use "sort" to do this with the following command line:

```
OS9: sort  < Phone_List  > Phone_List.sorted #12K
```

Note that the "<" and ">" are OS-9 redirection modifiers, not "brackets" around the filename. Find out more about the sort utility in the appendix.

Another excellent instance of an OS-9 utility is ar, available on most information services including Delphi and CompuServe. It produces a highly compact, or compressed, version of text files, ideal for transferring them via modem or storing them away for future use. I also make frequent use of LHA, a similar public-domain program available for downloading. Ar is very popular, but the .lzh files created by LHA are often more compact. LHA appears to be the compression utility of choice on some BBSes.

An excellent utility package called "Patch OS-9" is available from FARNA Systems or CoNect. This package of utilities, commands, and patches has been heralded as the next best thing to an updated version of OS-9 Level 2. In fact, it contains most of the updated utilities that "leaked out" from Tandy's aborted updating project. Other utility packages are available from Color Systems, Northern Exposure, C. Dekker, Sub-Etha Software, Hawksoft, and Bob van der Poel.

Telecommunications

There are plenty of **telecommunications** software packages (frequently called "terminal emulators") for OS-9. Each allows the user to call up an information service or bulletin board (BBS) to read and leave messages, upload and download programs, and so on. Each allows you to simultaneously use other OS-9 software. What a dramatic example of OS-9's multi-tasking power! Imagine downloading a long text file in one window while working with a spreadsheet and word processor in two others!

Commercial telecommunications software for the Color Computer 3 include "InfoXpress" by Bill Dickhaus. This fine package can be set up to automatically log onto a service (such as Delphi) or BBS then check for waiting e-mail and download new forum messages (in more than one forum). The user can then read and reply to their e-mail and any forum messages

while off-line, conserving on-line expenses and their time. The computer can be left on and set to dial, do its thing, then log-off totally unattended at any time of the day or night! Replies and new mail and forum messages are automatically uploaded the next time InfoXpress calls the same service. This fine program is available from Wittman Computer Products.

As you can see, each of the Big Five categories are well-represented In time, more and more OS-9 software will appear, ensuring that you will have little trouble finding what you need.

OS-9 Origins & Directions

OS-9: Origins, Directions

Even though your Color Computer purrs quietly in your warm home, the central processing unit (CPU) inside sharpened its claws and teeth in laboratories and factories around the world. Indeed, the personal computer — the most significant addition to home life since television — owes much of its usefulness to scientists solving mind-bending industrial problems.

Even today, computing standards set in the laboratory find their way onto the world's desktops. High resolution graphics and plenty of Random Access Memory (RAM), features typical of home computers such as your CoCo3, evolved from industry's advanced computer solutions. OS-9 on your Color Computer 3 also began as a solution to industrial computing needs.

Almost two decades ago technological and manufacturing giants were aching to have simultaneous control over several manufacturing and data collection processes. Motorola, a leading chip manufacturer, responded to the need by drawing up plans for a multi-tasking software development environment and advanced chip hardware. The chip was the Motorola 6809 (a chip from the same family is in your CoCo 3). The software development environment, from Microware Systems Corporation in Des Moines, Iowa, eventually evolved into OS-9. This is the story of how Motorola's and Microware's system — a sophisticated hardware/ software solution that is hard to beat even today — ended up on a desk sitting in front of you.

Motorola and Microware: Planning ahead for the 6809.

Microware's employees numbered only three in 1977. These three founders; Ken Kaplan, Larry Crane, and Robert Doggett, all remain with Microware in high positions. Their first project was a real-time executive (somewhat like the OS-9 Kernel) and a LISP and BASIC compiler. The expertise they displayed in these

products came to Motorola's attention and in late 1977 Motorola met with Microware to begin planning software for the forthcoming 6809 CPU. Their goal was to create a real-time executive and programming language for the new 6809 which would debut with the chip.

Motorola set stringent guidelines. First, the software should be designed to take full advantage of the 6809's new architecture. Second, the operating system, programming language, and the programs generated had to be ROMable — that is, programs should be able to he stored on an Erasable, Programmable Read-Only Memory (EPROM) chip. This program storage method is stable, compact, and easy to maintain in laboratory and industrial applications.

The chip's architecture incorporated an efficient, carefully optimized instruction set. It had a built-in multiply operator and addressing modes essential for multi-tasking operation. It was, as one Color Computer expert stated, "the best 8-bit CPU ever produced" (the 6809 has been classified by some as an "8/16-bit" CPU because it has some 16 bit capabilities). Microware knew that it was not enough to port existing 6800 software over to the (better) 6809. The new software had to be created from the ground up.

Microware finally delivered a state-of-the-art language, Basic09, and a real-time multi-tasking Kernel, OS-9. Together with the Motorola chip they formed an easy-to-learn, powerful system whose elegance and functionality remain well-respected around the world. To this day, not only are many older OS-9 (6809) based control systems still in operation, several companies still produce a wide assortment of controllers and modules.

A few details on Basic09

Basic09 is a Pascal-like, top-down programming environment with key words familiar to any BASIC programmer. The language also offers direct access to hardware. Thus, while its simplicity promised a softer learning curve, it still gave the programmer the control they required to control equipment.

Even more exciting, Basic09 was and may still be the fastest BASIC compiler on an eight-bit CPU. Basic09, included in your OS-9 Level 2 package, provides the pro-grammer a sophisti-cated programming environment worth hundreds of dollars (see Dale Puckett's article on Basic09 in the appendices).

As a testimony to BasicO9's speed, the famous BASIC programming benchmark, the Sieve of Eratosthenes, runs twice as fast on a 2MHz CoCo3 as it does under interpreted BASIC on a 8MHz Commodore Amiga, which has a much more advanced 16 bit 68000 CPU. Thus, the standard software and hardware included in CoCo 3 Basic09 programming is a better performer than the standard software and hardware Amiga users have for BASIC programming. A chip — as Motorola realized — is only as good as the software running on it.

The real-time Kernel handles the lower-level input and output (I/O). Written entirely in assembly language, the Kernel has an eleant, modular design that is extendable, maintainable, and very fast. The interrupt-handling allows quick response to changes in process states. Between the Kernel and Basic09, Microware had hit a home run with its new package.

The birth of OS-9

The Des Moines-based software company, with both software elegance and marketing strategy in mind, didn't stop there. Andy Ball, Microware's marketing director, said recently, "We didn't want just a real-time Kernel, we wanted to provide complete support." This support arrived with Microware's OS-9, a complete operating system for the 6809 which included a UNIX-like Shell, a Kernel, I/O routines, and Basic09. It later added a
UNIX-compatible C compiler that provides a degree of portability and dependability not found on most C compilers.

While OS-9's similarity to UNIX is widely noted, Ball makes it clear that Microware had its own design goals. UNIX itself was relatively new when OS-9 was developed so its influence is much less than most observers suspect. "OS-9 isn't trying to compete with UNIX," explained Ball. "UNIX is really the best in its class, and we like to think that OS-9 is the best in its class." As an indication that Microware complements and does not compete with UNIX, Ball points to OS-9's powerful UNIX-compatible languages, especially the Microware C language. This makes OS-9 an ideal environment for creating UNIX-compatible applications.

One distinction is that OS-9 is "real-time"; that is, "it can respond to external events as they occur," says Ball. He comments further that while UNIX can be "real-time" in certain applications, OS-9 generally performs much faster. UNIX, developed on mainframes that deal with user problems in "batch" fashion, is typically less responsive. Also, UNIX

and its software is often not "re-entrant", necessitating multiple copies of the same program in memory. This unnecessarily hogs system resources and ultimately the UNIX-based computer loses speed. OS-9 is re-entrant and thus much kinder to system resources, which means it can do the same job as larger systems with much less. This is one reason typical OS-9 programs aren't nearly as big as similar MS-DOS (and other systems) counterparts.

Microware also designed OS-9 to be modular. Software engineers enjoy maintaining modular systems. Trouble with the printer module? Replace it with another! Since Motorola built position-independent code (PIC) capability into the 6809, these modules can be loaded in at just about anytime, just about anywhere in memory. Adding a new device driver or application program is as easy as loading it from disk. The versatility of OS-9's memory modules distinguishes it from any other major operating system.

The Tandy Connection

OS-9's marvelous features did not go unnoticed. When Tandy heard that many CoCo owners were looking for more powerful software, they examined available operating systems for the 6809E and chose OS-9, not just for the multi-tasking capabilities but for the application software available under OS-9. This proved to be a clever choice that in many ways made the Color Computer 3 possible; without OS-9, the Color Computer as we know it may well have "passed away" much earlier.

Choosing OS-9 for the Color Computer perhaps demonstrated another sign of Tandy's innovation. Tandy Corporation had long been an innovator in the computer field, a fact which the media and the public has been slow in acknowledging. "Tandy deserves a lot more credit than it usually gets," Ball comments. "They produced the first (really successful) personal computer [TRS-80 Model 1 in 1977] and the first laptop [TRS-80 Model 100, 1983]."

Unfortunately, Tandy has been forced by today's highly competitive computer market to "go along with the crowd" and offer a line of IBM compatible clones. Yet they still manage to be innovative; they were among the first computer retailers to mass-market multi-media computers.

"Tandy has more than 9% of the computer market, and their most popular computer [in 1988] is the Color Computer," Ball continues. "The CoCo is really the main component of their

market." Since the introduction of OS-9 on the first Color Computer, OS-9 was a strong seller on the CoCo 2 and, of course, on the CoCo 3 as OS-9 Level 2. Tandy announced near the end of the CoCo's long life that no disk software would be marketed unless it was OS-9 Level 2-based. Level 2 had finally become the "official" operating system for the CoCo 3.

Microware decided to agree on a Color Computer port of OS-9 based on the observation that the Color Computer OS-9 system would be a low-cost platform to demonstrate OS-9's features to the public at large. In fact, Microware had contacted an independant firm (Frank Hogg Labs) about porting OS-9 to the CoCo even before getting Tandy's attention. The system proved so powerful that even today engineers working with some of the most powerful mainframes develop software at home on a Tandy Color Computer running OS-9's C compiler. This software, perfected at home, will run without a hitch on their mainframes at work!

Ball himself was impressed by the Color Computer 3's price/ performance ratio. "No other 8-bit computer can give you 512K of RAM, a Kernighan and Ritchie C compiler, and performance that the CoCo 3 provides. There just is no comparison."

OS-9 Today

Microware has not ended its relationship with advanced Motorola chips. Although the bulk of its work on 6809 OS-9 was completed by 1983, the advent of the Motorola 68000 offered them a chance to develop the next generation of OS-9. Called OS-9/68000 or simply OSK by some, this version of OS-9 is offered on a large number of 680x0-based machines.

Most OSK systems are not mainstream personal computers. Software developers use them to develop software for other machines — even when the target machine is not 680x0 based. Companies from the US, Germany, England, and Japan offer OSK systems whose power and utility continues to increase with Microware's UniBridge and SLED (a unique source-level C language debugger).

OSK continues to prevail in process-control. In fact, its elegance
and versatility in this arena drew the Midas gaze of Philips and Sony, both mass-market electronics leaders, in their search for an advanced operating system for Compact-Disk interactive

Compact Disk Interactive (CD-i), an interactive computer

entertainment and education system, is poised ready to revolutionize home entertainment and learning. CD-RTOS, its operating system, has OS-9/68000 as its heart. [See the Essay on CD-i for more insights into this new technology.]

Other industry leaders attracted to OS-9 include Honeywell, NASA, Hughes, and Boeing. Some of the applications involving OS-9 are secret or proprietary. Some that aren't: the systems aboard a Swedish-made Corvette-class naval ship (a bit smaller than a destroyer) are all controlled by OS-9, as are many Space Shuttle-related projects. OS-9 is used extensively in cell-controllers which collect essential data. It is used in testing units at automobile manufacturing plants and electronics testing firms. Rumor has it that OS-9 is used in some on-board systems monitoring equipment in new cars. So if your oil-light comes on and saves your car from melt-down, you may be able to thank OS-9.

In the personal computing world, the most exciting news for many OS-9 users arrived in OSK for the Atari ST series. The ST port, begun by a now-defunct operation in California and saved by Microware, is currently available from Cumana Ltd., of England. The Atari ST, though crippled with a few design flaws [see the Essay on OS-9 and Music] provides a lot of "bang for the buck" under OS-9. Microware used STs to aid in CD-i software development. One insider described the ST software Microware created for CD-i as "spectacular".

OS-9 is now available for the Commodore Amiga. The port, by Tesseract Software of Australia, is not commercially available in the United States but can be ordered directly from the company. It is pricey at around $600.00 US though.

Kindled interest in the U.S., popular ity abroad

While OS-9's popularity grows in the United States, OS-9 "fanaticism" is rampant in Japan where OS-9 is the number one business operating system. Microware has 12 full-time employees in its Tokyo office from whom reports filter back about OS-9's competitors: the competition advertises that they offer "OS-9-1ike operating systems"! Its popularity is rising in Germany, too, perhaps as a result of the success of the Atari ST which is the German's first choice in business computers. Microware also has an office in the United Kingdom.

Is OS-9 in everyone's future?

Soon, a derivative of OSK, used in interactive television set-top boxes, could well be in more homes than MS-DOS, UNIX, and System 7.x (Apple Macintosh) combined. Still, Microware has a

challenge. In the years ahead newer, better equipment than even that found in CD-i and set-top TV boxes will be widely available — beginning with the next generation of microprocessor chips from Motorola, IBM, and Apple.

These chips, the RISC based PowerPC series, offer parallel processing among other hardware achievements. Parallel processing speeds up computing by breaking problems down into small parts, each of which is tackled simultaneously by different CPUs which are monitored by a "master" CPU. This provides a form of "real" multi-tasking which operating systems like UNIX, OS-9, and OS/2 only simulate. Processing power such as this will indeed revolutionize the way we work with computers. OS-9 (in the form of OS-9000, which is written in C for portability) is now available for new chip series, and also for the popular Intel 80x86 series.

The future of the CoCo

The Future of the Color Computer

We, the current Color Computer users,
ARE our own future. As long as we continue
to use and support these fantastic machines,
there will always be new software
and hardware available.

So support those small vendors and
the core of excellent programmers and
writers out there who are still willing
to support you. In many ways the CoCo,
especially when paired with OS-9,
was ahead of its time.

And, as evidenced by a strong
(though small) support community,
its time is far from passed...

Do I really have to learn hexadecimal?

Day Nine: Hexadecimal numbers and Boolean Algebra

A great mathematician and logician brought computer science to dizzying heights with Boolean algebra, a field of mathematics involving binary (base 2) arithmetic (in Boolean alebra, 1 + 1 = 10). Boole's contributions left a great number of people staring blank-eyed at his dizzying height and wondering why he was up there. After all, who cared that 0 is false and 1 is true, and that if you made true false and false ture, it wouldn't change a thing? Computers cared.

Bits and Bytes

Binary arithmetic and truth tables may leave your mouth dry but computers drool over them. Binary arithmetic has a direct correlation with the hardware of which computers are made. In computer applications, though, "true" and "false" correspond to "on" and "off". If a switch is on, a condition is true. If a switch is off, a condition is false.

Feeding a computer a series of instructions amounts to feeding it 0'sand 1's. Computers are happy to eat these, too; one of these binary digits is called a bit (Binary diglT). Four bits is a nibble; two nibbles a byte. You might think that two bytes is almost a meal, but actually it's more often called a word.

Although computers eat bits and bytes for breakfast, humans don't. Most of us aren't even sure that bits and bytes can be numbers. Quick, what is 10010011? The number is 147 by our usual method of counting, called decimal.

If you feel adventurous, if you like to solve puzzles, and if you ever wondered how written words relate to what they represent, read on. Still with us? Great.

Binary numbers

In order to keep our numbers straight in this essay, let's agree to precede all decimal numbers with "#", all binary numbers with "%", and all hex numbers with "$".

In binary, you are only allowed two characters, 0 and 1. lf you have nothing you write, "I have 0." If you add one you write, "I have 1." Add another and, well, what do you do? You "know" you have two things, but how do you write it? Take the one and move it over to your left side and pretend that, when the one is over on the left, it really means "two".

OK, so %10 stands for two (decimal). You won't be surprised that Col 1 is three (decimal) — just add 10 + 1 = 11. What is four? If you guessed that four is written by moving a 1 a little

farther to the left, you've caught on. Four is %100 — the two zeroes tell us that the one is over two places to the left and thus represents the next power of two, namely 2 X 2 or four. Move the one over to the left one more place, it represents eight.

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
16	10000
32	100000

This is EXACTLY the same process we use for decimal notation. If y ou have one, write 1. lf you have ten, move the 1 to the left and now we pretend it means ten. To remind you that the one is "to the left" we stick a zero to the right. Ten is then "10". Each time we add a zero to the right of a 1, we pretend that the "1" represents a higher power of ten. Thus, "10" is ten, "100" is one hundred, and so on.

Writing down numbers — which are really abstract ideas — raises interesting questions about the relationship between ideas and writing. It shows how our culture quietly agrees that a symbol — for example, the numeral "2" — becomes the same idea for many people.

But a word only represents an idea and therefore exists in a domain separate from the letters and numerals we use to express it. Different cultures have different words for similar ideas. In the same way, "2" and "10" are different symbols for the same idea: two.

A little practice

In decimal, 10 is ten. In binary, %10 is two. In decimal, 100 is one hundred. In binary, %100 is four. And, in decimal 10,000 is ten-thousand. In binary, you write ten thousand%1001 1100010000. Got it?

The good news is that MOST OF YOU DON'T EVER HAVE TO LEARN BINARY TO DECIMAL CONVERSION.

Hex notation: compact and easy

There are two chief reasons why software engineers use hex notation. First, hex notation is more compact than decimal notation. The decimal number #49,876 takes up five digits; in hex it takes up four ($C2D4). Second, and more importantly, converting between binary and hex is a snap. Hex is created by counting powers of 16, binary by counting powers of 2. Since 16 is a power of 2, hex is closely related to binary.

But still, why prefer it over decimal notation on which we were all raised? To see why, let's examine counting.

Counting in hex

Imagine two planets, Terra One and Terra Two. Both are Terratorial and Terra-fied of each other because inhabitants of Terra One have ten fingers while Terra Two-ites have sixteen. Plus, there are political differences; Terra One is fiscally conservative while Terra Two just loves to distribute money liberally among its population.

One day, a professional coin-counter specializing in pennies assembles in the square with an accountant known for penny-pinching. The coin-counter commences counting: "One two three four five six seven eight nine ten. OK, that's one dime."

Notice that he stopped at ten. They start counting the second dime: "One two three four five..." and so on, really meaning they have ONE dime and ONE penny, ONE dime and TWO pennies, and so on. After the coin-counter counts each penny, the accountant would write in his ledger:

DIMES	PENNIES	
0	1	
0	2	
0	3	
0	4	
0	5	
0	6	
0	7	
0	8	
0	9	

"OK, that's one dime."

1	1	
1	2	and so on.

Back on Terra Two...

Switching scenes to Terra Two we find another accountant with another coin-counter. Ledger in hand, the accountant observes his sixteen-fingered friend count a pile of pennies. “One two three four five six seven eight nine A B C D E F. OK, that’s one dime. One two three four ....” We, as cultural historians, realize that A through F are required because our sixteen-fingered friends from Terra Two don’t have enough digits in their numbering system. Zero through F translates to 0 through 15 on Terra One. The accountant, his sixteen fingers fumbling with the pencil, dutifully writes:

DIMES	PENNIES
0	1
0	2
0	3
0	4
0	5
0	6
0	7
0	8
0	9
0	A
0	B
0	C
0	D
0	E
0	F

“OK, that’s one dime.”

1	1	
1	2	
1	3	and so on.

Also, as cultural historians, we know that if a person from Terra Two offers us a dime we take it. That dime is worth sixteen cents. Whatever you do don’t trade it for a dime from Terra One!

Any technology sufficiently advanced, it is said, becomes magic; any civilization sufficiently advanced becomes a welfare state.

The scene on Terra One:

And so it is on Terra One and Two.

A welfare worker, given 100 pennies (equivalent to 10 dimes) by the Big Banker to distribute among welfare recipients, has just arrived at the Welfare office:

Welfare Worker (over anxious crowd noise): "OK I have 100 pennies here. I have to split these up among the federal welfare recipients, the state welfare recipients, the county recipients, and the local recipients."

Subversive Welfare Recipient: "Exactly how do you expect to divide the 20 dimes?"

Welfare Worker: "I'm willing to listen to suggestions."

Recipient: "Why don't you give half your pile to the federal recipients, half the remaining pile to the state recipients, and so on until all the pennies are gone?"

Welfare Worker: "Why not? Sure. Let's see. I have 100 pennies. OK, fifty go to you (gives 50 cents to federal recipient) and that leaves me with 50 pennies. Half of that — twenty five — go to you (gives 25 cents to the state recipient). Wait... does anybody have change?"

Welfare worker gulps, suddenly realizing one can't split 25 cents in equal halves. County and local recipients, obviously not recipients today, begin rioting.

Also on Terrra Two

The same thing is going on on Terra Two:
Welfare Worker: "OK, today we are going to distribute our money as our friends on Terra One do it — halving the pile each time."

Another Subversive Recipient: "I'd like to see THAT ..."

Welfare Worker: "I have 10 dimes. Let me get change — OK, our dime has sixteen pennies so I now have what Terra One dwellers would call 10 times 16 pennies, or #160 pennies. Half to you and that leaves #80; half to you and that leaves #40; half to you and that leaves #20; half to you and that leaves #10."

Everyone goes home happy. The moral: Sixteen is a power of two because it can be written as 2 X 2 X 2 X 2. This is the same as saying that 16 = 24. Because base 16 (hex) is founded on a power of two, Terra Two can expect fewer riots. And because hex is based on a power of two, hex numbers are easily divided by two. Dividing a hex number by two is essential in converting a hex number to binary. Thus, it is easy to convert hex numbers into binary numbers.

Example: Note that sixteen, as a power of two, is nicely represented in both binary and hex. In binary, it-is %10000. In hex, it is $10. Let's convert $32 into binary:

$32 = $30 + 2
= 3 X $10 + 2

Now, 3 in binary is 11 and $10 (hex) in binary is 10000. Thus, 3 X $10 is the same as 11 X 10000 in binary. This multiplication yields 110000. Also, $2 in binary is 10.

Summarizing:

```
   00000010 =          $2
+  00110000 = + $3 x $10
   00110000 =          $32
```

Notice that adding binary numbers in columns is easy. Just remember that 1+0 is 1, 0+1 is 1, and 1+1 is 0 carry a one. This is just like ordinary decimal addition except that instead of counting on your fingers, you count on your kneecaps.

Converting Decimal to Hex

Now that you appreciate why computer scientists like hex (well, perhaps you still don't), we raise the harder question: how do you convert between decimal and hex? Easy; take a decimal number and find the largest power of sixteen that is just smaller than that number. Subtract. Do the same with the remainder over and over until you are left with just numbers in the "ones" place. Add all those powers of sixteen in hex notation and top it off with the remainder in the ones place and you're done.

Those instructions resemble one of Tom Lehrer's dry comedy classics. They also resemble math torture. The good news is that MOST OF YOU DON'T EVER HAVE TO LEARN HEX TO DECIMAL CONVERSION. No, you don't have to learn hex conversion.

If computers become your passion chances are good you'll learn how to convert hex numbers on sight. You will win the admiration of many. But many of you don't want to take the time. Just for you, the "Mastering OS-9"' disk provides a utility called Convert.

the **convert** *utility*

Convert is a Basic09 program which incorporates one of Stephen Goldberg's excellent utilities. It converts between decimal, hex, and binary numbers. Just input a number preceded by a special symbol. Depending on the symbol the program will

read the number as hex, decimal, or binary. The program then converts that number into all three forms.
Convert requires that you have the "Runb" module in your current execution directory. To check, type:

OS9: dir x <ENTER>

This provides a directory listing of your current execution directory. If neither Convert nor Runb are in memory, they should be listed in this directory.

Example: Upon running convert you will see:

```
**************************************************
              Welcome to Convert

Provides Decimal, Hex, and Binary versions of input
number
**************************************************

             Press <BREAK> to Exit

Press <$> for Hex numbers    or    <#> for Decimal
              then type the number
```

Then you might type: **$35 <ENTER>**. This is hex 35. The program then responds:

```
53 $35 %00110101
```

This tells you that $35 is 53 in decimal notation and 00110101 in
binary. Then you might type: **#23 <ENTER>**. This is decimal 23. You would see on your screen:

```
23 $17 %00010111
```

If you typed: **%10 <ENTER>**, you would see:

```
2 $02 %00000010
```

Press the <BREAK> key to exit. You will then see:

```
(c) 1988 Kenneth Leigh Enterprises
Portions (c) 1987 Stephen B. Goldberg and FBN Software
Used with Pernission
```

An easier waytto convert hex/binary/ decimal numbers

There is an easier way to calculate hex/decimal/binary conversions by hand. All you have to do is remember that each hex character represents a value from 0 to 16. So break each hex number into “nibbles” (single characters) and try to remember the following 32 values:

DECIMAL	HEX	BINARY	DECIMAL	HEX	BINARY
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

Take the hex number $F2. The two nibbles are “F” and “2”. “F” in binary is 1111 and “2” is 0010. So the binary equivalent is 11110010. Converting from binary to hex is the same... only in reverse. Should be easy enough!

Converting a decimal number is a bit more work. You convert it to a binary number first. To do this, you start writing digits from left to right. If your decimal number is 128 or more, it is an eight bit number. The left-most posiotion represents 128. From left to right, the positions represent 128, 64, 32, 16, 8, 4, 2, and 1. A “1” in a position means the number is greater than or equal to that position's value, a “0” means it is less.

Let's convert 242 into a binary number. Since 242 is greater than 128, we start by writing a “1”. Subtract 128 from 242. That leaves us with 114. Since 114 is greater than 64 we again write a “1”. Subtract 64 from 114, leaving 50, which is greater than 32, so another “1”. Subtracting 32 from 50 leaves us with 18. So another “1”. 16 from 18 leaves 2. Since 2 is less than 8, there are no “eights”, so we write a “0”. Same for the “4” position. 2

IS 2, so we have a"1" there, and since there are no "ones", another zero. So our binary number is 11110010.

CD-i: Compact Disk Interactive

To concert a decimal into a hexadecimal number, convert it to a binary first. Now break the binary number into two nibbles (four digit numbers). Using 242, this would be 1111 0010. Now look at our chart and write down the values. 1111=F, and 0010=2.

Hex numbers are not scary, just tedious. Software and hardware engineers find them useful; if you plan to follow these careers, learn hex. Otherwise, learn to use convert.

Compact Disk-Interactive: OS-9 in Every Pot

I poped my head into a video arcade several years ago (around 1985) to see a huge crowd aching to see the newest video game. It was "Dragon's Lair", an interactive video disk game with excellent cartoon-like animation that rivaled broadcast-quality television (this is now considered a classic!). I saw the future that day.

Then came Compact Disk players. CD Digital Audio was indeed music to my ears. The first demo disks sported some of Japan's best jazz musicians, but my favorite sound was the one I heard right before the music started playing: nothing. No static from a record stylus, no tape hiss... absolutely nothing! I heard the future that day.

And then came CD-ROM. Using compact disk technology to store error-free data — not just millions of bytes, but HUNDREDS OF MILLIONS of bytes of data (usually 650 megabytes + per CD) — was incredible! Think of the birth records, property deeds, and scientific data you could store and quickly recall using database management programs! Think of the reference works! You cold put an entire encyclopedia, dictionary, and thesaurus on a single disk. True, you had to have a computer with the necessary software to display and use the information, so what? I knew the future when I saw it! (Editor's note: Remember, this was originally written in 1988, long before CD-ROM was affordable or even practical!)

Well, it was the future. The three-headed dragon of CD Digital Audio, CD-ROM, and Video Disk-Interactive has been slain by Microware, Sony, and Phillips. Together these companies have created Compact Disk-Interactive (CD-i), which combines most of the virtues and few of the weaknesses of these three for-mats. The result? An interactive, informative, and entertaining audio-visual marvel you connect to your TV and (optionally) your

Another TV gadget?

stereo. No personal computer is required because the computer is built-in. No special software to load because each disk is self contained with all data and programs needed to operate its title, and at a reasonable cost.

Imagine walking into your living room or wherever you keep your TV, turning on your CD-i player and popping in a disk. Turn on your super high-resolution stereo television and see colorful full-action video followed by a short animated sequence. During the animation glorious full stereo music pours out of your speakers. Finally, a screen appears: "Welcome to Moon Flight, Press Remote Control Now"... and it says this too. With iced tea in hand and relaxing comfortably in your recliner, you activate the show with your remote.

Off you go into an hour long trip to the moon. At various points in the program you can stop to look back at earth — choose just what time of day you want it to be; look up at the stars; or examine the moon. Just touch the proper button on your remote as displayed on the screen (and sometimes spoken to you!). Depending on the heavenly body you wish to examine, you are given choices on screen to get more information about what you are currently looking at. What is that city down there? Isn't that the Great Wall of China I see? Hey, let's take a side trip to China! Just push the proper remote button for more information...

Fun to use, hard to create

No, the CD-i player may LOOK like another TV/Stereo gadget, but it is much more. In reality, it provides entertainment and educational opportunities in a highly advanced computer. This is the start of the "computer appliance" so many people have envisioned for years. So easy to use a child can easily operate it alone. No more intimidating than a stereo tape player, VCR or any number of other now common gadgets.

The entertainment value of CD-i is tremendous. But creation of interactive disks may makes this medium a product of expert programmers... more usually prgramming teams. How are these disks assembled? I t requires personnel coordintation of night-marish complexity. Programmers, video producers, composers, and graphics designers all meet with one goal in mind: they want to create a single disk to entertain and inform their target audience based on the best each has to offer.

Conceiving such an enormous audio-visual project is a delicate task for a director. Combining text, video, animation, and audio

onto one disk isa delicate task for a programmer. And coughing up a quarter million dollars (sometimes more) for each title is the unenviable task for the producer. Some of the first titles may have cost millions to produce! With some of the modern technology available today, it IS possible to cut costs dramatically. It isn't inconceivable for a couple people (programmer and video/animation specialist) to be able to produce a CD-i title. Not as long as they have the $80-$100,000 of equipment needed.

How CD-i works

But the pay-off is there. While predictions are still cautious, the public is starting to accept this revolutionary medium. All of the popular video game machines are now using CD-ROM for ever larger and realistic games, and it is almost mandatory for computers to have CD-ROM drives. As the public gets more familiar with these devices, they will want more from them, and those who don't have access to computers, or feel intimidated by them, will probably feel more at home with a CD-i player. Cost is more justifiable too... nothing else on the market today can play audio CDs, games and educational software, and full length movies (Panasonic promises as much with its rival "3DO" player, but this machine uses a different format and cannot yet play movies).

Why OS-9 for CD-i?

An interactive disk contains a variety of data types. There are several audio formats, several video formats, and text formats offering a wide variety of fonts. Animation is supported in many forms and common video effects (mosaic effects, wipes, and chroma-key type techniques, for example) are available. Audio can range from AM radio quality all the way to CD digital audio with many intermediate formats. Programs and their data are also included on the disk to activate animation and control what the viewer is presented with.

This data is interleaved on the disk in such a way that all elements can be synchronized during playback. Data is frequently encoded in compressed form on the disk. This interleaved data is read by the players heads at a constant rate and sent by the operating system, CD-RTOS, to the proper decoding hardware. When data has been decoded it is then queued up
for presentation to you.

When Sony and Philips conceived CD-i, they knew that the software to run it would need to be easily configurable to the specially-designed hardware; it would need to react in real-time to various inputs and events, it would have to incorporate a

sophisticated and reliable user interface and programming environment, and it would have to be multi-tasking.

This should sound familiar. You probably won't be surprised to learn that OS-9 is the heart of CD-RTOS. To be precise, OS-9/68000 is the heart of it. CD-RTOS contains the Kernel of OS-9/68K and includes new manager and driver software to accomodate the state-of-the-art hardware added by Sony and Philips to the CD-i player.

One of the greatest achievements of CD-i is its synchronization of different data output. OS-9 is a natural for this kind of work as it both multi-tasks and responds to software signals with sensitivity. (Perhaps"sensitivity" is the wrong word; in computerese this translates into the ability to handle several different levels of interrupts.) CD-RTOS, being modular, also allows software developers to write replacement modules which fine-tune the CD-i player for a particular disk. This is much like the techniques Color Computer 3 users employ to add some pep to their OS-9 based computer.

OS-9 in every home...

The CD-i insiders interviewed for "Mastering OS-9" have high hopes that CD-i will take off like wildfire. This will be true if the quality of the disk programming is high. As great a challenge as assembling the hardware was, the greatest challenge is still ahead: make it entertain without seeming gimmicky or shallow. Whether or not media leaders want to invest millions in title development remains to be seen. How quickly they jump into the market may determine the whole future of CD-i.

One side-effect of CD-i's success is that OS-9 may be in more homes than any other operating system in less than two years. "But," you may object, "it's only OS-9 strapped into a piece of stereo equipment!" Not so. As reported in "Compact Disk-lnteractive: A Designer'sOverview" (McGraw-Hill, 1988), the bulk of OS-9/68K software will run virtually unchanged on a CD-i player (within the hardware limits of the player itself). This means that with a little additional hardware (a keyboard and disk drive, or perhaps a terminal), thousands could have a state-of-the-art, multi-user, multi-tasking microcomputer in their homes. If the demand to use a CD-i player as a computer increases, the availability of OS-9/68K software might explode. Some of this OS-9 excitement is sure to trickle down to Color Computer users.

Advantages of CD-i over other "game" machines

The main advantage the CD-i machine has over the other game

machines is the ability to play feature length movies. Not only is the quality of digital video better than tape, but you can do more with it! I recently saw a demo of a CD-i machine playing "Star Trek IV". The disk has several "chapters" stored on it. One can bring up a menu that shows a still scene from a certain part of the movie. From that menu, it is easy to jump directly to that scene in a matter of seconds! No more fast-forwarding and trying to watch jumpy video so you can get back to where you stopped watching a day or two ago. And it is really easy to get close to your favorite part! Although some of the other machines, namely Panasonic's 3DO, have advertised movies as being available, this feature has yet to materialize. So the CD-i palyer is the most versatile piece of entertainment equipment available. Stereo CD player, video disk, game disk, and educational programs too!

More info on CD-i

For more information you can order "Compact Disk-lnteractive: A Designer's Overview" directly from McGraw-Hill. Written by Philips, it provides a sketch of what CD-i titles will look like, how they will be created, and how the hardware works. There is a generous sampling of technical information. CD-RTOS is also explained. You may also write Microware Systems Corporation:

Where to find CD-i players

New Media
Microware Systems Corp.
1900 NW 114th Street
Des Moines, IA 50322.

If you just want to see on of these amazing machines, they are as near as your closest Sears, Macy's, Babbage's Software, or Best Buy stores. With the support of these wide-spread retailers, new titles are in most malls across the country. And if all else fails, telephone Sears.

CD-i marketing... off target?

None of the stores are pushing these devices. One problem is that store personell just aren't very knowledgeable. Another is the way they appear to be selling them.

The Phillips CD-i machine is being sold primarily as a family entertainment/educational product. There are many game titles, but the emphasis seems to be toward adults, who will but more for educational purposes. This is in direct contrast with Panasonic's similar 3DO marketing. Panasonic is emphasizing game play, in competition with the Sega CD and similar game systesm, then tacking on education as a nice by-product. Unfortunately, this may be working. Kids see these, then talk parents into getting them. Maybe Phillips should shift their emphasis on game play also, then mention the multi-purpose and educational qualities as well.

Tutorial 7: What we'll cover

GETTING STARTED WITH TUTORIAL 7

OS9Gen

The first part of "Getting Started With Tutorial 7" is intended to be a stand-alone guide to creating a custom boot disk with OS9Gen. Instead of separating background information on OS9Gen from step-by-step instructions in using it, "Getting Started With Tutorial 7" includes in one essay what you need to know to create a disk with your favorite modules and commands.

Why change the book's format just for this one utility? Simply because you will probably refer to this essay on OS9Gen more often than any other section of "Mastering OS-9". By placing all the relevant information in one place you won't have to wade through numbered steps.

The second part of "Getting Started With Tutorial 7" formally introduces you to one of the truly revolutionary aspects of Level 2 for the Color Computer 3 — multi-tasking windows. The hands-on tutorial following this introductory material ONLY applies to creating these windows.

First-time readers of "Mastering OS-9" are expected to perform all steps associated with OS9Gen in the essay below so don't skim it lightly. The bootdisk you build is required for Tutorial 8.

OS9Gen: *into the guts of a boot*

OS9Gen provides the greatest user interaction with the boot-making process. Use OS9Gen whenever you create a new boot disk from scratch and particularly when adding modules to OS9Boot which are not included on the System Master distribution disk.

OS9Gen creates a new boot file on a freshly formatted disk. It only includes modules listed in a specified bootlist. In this way it is similar to config. The difference is that config automatically creates its own bootlist based on some choices you make. OS9Gen requires you to use edit (or any other text editor) to create your own bootlist. You could, of course, modify an existing bootlist, such as the one made by config, instead of creating a new one. In this tutorial, we'll start from scratch.

A bootlist contains module names, one per line. Recall from Tutorial 1 that OS-9 Level 2 can be sensitive to the order in which these module names appear in the bootlist. Thus, Config — which takes bootlist creation out of your hands — has limitations. OS9Gen overcomes these.

A bootlist used with OS9Gen is usually situated within a directory containing the actual modules listed in the bootlist. This directory is commonly called MODULES or MODS. This directory with its bootlist and modules will reside in drive /d0 for this tutorial.

Before creating the new boot on drive /d1 you must format a fresh disk. Then create the bootlist with a text editor, saving it in a modules directory. Finally, use OS9Gen to create a new boot disk's OS9Bootfile.

This done, copy over to the new boot disk a CMDS and SYS directory and a Startup file using dsave and the Shell's pipe modifier. We demonstrate dsave here but we wait for a more complete introduction to this powerful command until Tutorial 8.

OS9Gen *syntax*

OS9Gen works with the following syntax:

OS9: os9gen device < bootlist <ENTER>

Here, device is the destination drive (see Tutorial 1) for the OS9Boot you create. While any valid floppy drive may be the destination drive, we will use drive /d1.

Bootlist refers to the name of a text file you create containing module names, one per line, for each module to be included in OS9Boot.

Bootlist can reside anywhere, although it is usually in the current data directory. lf it is not, use a more complete path name to help OS-9 find the file when running OS9Gen. Also, the module names listed in bootlist may be complete path names for the modules if these modules do not reside in the current data directory.

The "<" symbol indicates that OS9Gen reads bootlist line by line as its input. OS9Gen takes each module name it finds, looks for a module with the given path name, and places that module in the new boot.

Other path redrirection techniques

There are other alternatives, especially with the utilities included with "Mastering OS-9". One utility, from Steve Goldberg, is D and provides a line-by-line directory listing of the current data directory.

If all the modules for your new boot are contained in your current
data directory, you can run OS9Gen like so:

OS9: d!os9gen /d1 <ENTER>

The trouble with this method comes when the modules in your current directory must fall in a particular order in the OS9Boot file. Some hardware and software combinations on a system may require different boot list orders. Once you find a boot list that works, however, you can transfer the modules named in it over to a MODULES directory of their own. Then you can use the above command line with no problems.

If the output of D does not create a workable boot list, you have encountered the "Bootlist Order Bug" (commonly referred to as the "BLOB"). This bug has been carefully researched by OS-9 specialists around the country. There is a detailed description of fixes in the appendix.

First step: a modules directory

I recommend that a backup copy of the OS-9 Boot/Config disk be used to store all modules. This disk contains a directory named MODULES already created for you by Tandy. It contains all standard Tandy modules. "Mastering OS-9" brings you four additional modules thanks to Bill Brady and Kevin Darling. Any third-party modules (that is, modules from vendors other than Tandy) should be copied into this directory as soon as you get them. If you have "Patch OS-9" from FARNA Systems or CoNect, you will want to copy any additional modules to your "Bootmaker" disk instead. Whenever you add modures to the MODULES directory, backup the disk.

Example: Suppose you buy a house alarm system to run off your OS-9 based Color Computer 3. The distribution software arriving with it comes with devlce drivers and descriptors for the alarm sensors. Suppose further that the device driver and descriptor are alarm.dr and alarm.dd respectively and that the manufacturer places these in a MODULES directory on the distribution disk. Place this disk in drive /d0. Place your Boot/Config backup (or Bootmaker) in drive /d1. Now type the following:

OS9: chd /d0/MODULES <ENTER>
OS9: copy alarm.dr /d1/modules/alarm.dr <ENTER>
OS9: copy alarm.dd /d1/modules/alarm.dd <ENTER>

Now that all the modules you need are in MODULES on the Boot/Config backup disk, you must create a list of each module name in a text file. One way to create such a list is to use edit or some other text editor to modify the bootlist already in the MODULES directory in the Boot/Config disk. Or you can use the "Mastering OS-9" D utility and send its output to a file which you can easily edit with a full-featured word processor. Let's start from scratch. Here is a bootlist which is known to work:

```
os9p2
init
ioman
RBF.mn
cc3disk.dr
r0.dd
rammer.dr
d0_40d.dd
d1_40d.dd
ddd0_40d.dd
SCF.mn
cc3io.dr
grfint.io
term_win.dt
w.dw
w2.d
w3.d
w4.d
w5.d
w6.d
w7.d
pipeman.mn
piper.dr
pipe.dd
mw
p.dd
proacia.dr
printer.dr
p.dd
clock.60hz
cc3go
```

MWP.dd and PROACIA.dr- are are the names of public-domain modules which permit telecommunicating with Bill Brady's shareware program WizPro. Once you create a boot disk with

the above modules you can run WizPro the first day you receive or download it. Check an information service such as Genie, Delphi, or CompuServe for more details. Of course, if you won't be running WizPro these aren't needed.

Many people replace Tandy's stock ACIA.dr with SACIA.dr, which is an enhanced public-domain version. This is just one of the patches found on the "Patch OS-9" disk set. It can also be found on many bulletin boards that support OS-9.

The modules r0.dd and rammer.dr install a ramdisk on your system. You use this ramdisk for the first time in Tutorial 9.

Create the list above with a text processor such as edit or use your own word processor. Make sure you save a copy of the resulting bootlist in the MODULES directory. For the sake of this tutorial, call the bootlist "bootlist.sO9".

Running OS9Gen

Put your backup of the Boot/Config disk in drive /d0. Now type the following commands, pausing between them to follow our additional directions:

OS9: chd /d0/modules <ENTER>

Check to make sure your bootlist is in the MODULES directory:

OS9:dir <ENTER>

Do you see Bootlist.s09? lf so, fine. If not, copy it into MODULES:

OS9: chx /d0/cmds <ENTER>

Confirm that OS9Gen is in your current execution directory:

OS9:dir x <ENTER>

It should be there if you're using the Boot/Config backup. If it is not, find it and either copy it into your current execution directory or load it into memory.

Place a freshly formatted disk in Drive /d 1 (for help with the format command, see Tutorial 1) . Type:

OS9: os9gen /d1 < bootlist.sO9 <ENTER>

Everything is automatic. OS9Gen will check the disk in drive /d1 to make sure it is formatted and a valid OS-9 disk. It will then read bootlist.s09 line by line and merge the modules into a new OS0Boot file which is placed on drive /d1.

What next?

At this point, I suggest finding a system disk which includes your favorite commands and procedure files. Place that disk in drive /d0.

If you are reading "Getting Started With Tutorial 7" as part of

the “Mastering OS-9” tutorials, please use the Custom Master #1 disk as the system disk to be placed in drive/d0.
Orient OS-9 to the new system disk by typing:

OS9: chd /d0; chx /d0/cmds <ENTER>

Check that dsave and makdir are in CMDS. If not, find them on another disk and load them into memory:

OS9: dir x <ENTER>
OS9: load dsave <ENTER>
OS9: load makdir <ENTER>

Run dsave and pipe its output to a shell for immediate execution:

OS9: dsave /d0 /d1 ! shell <ENTER>

sit back and watch dsave do its thing. Users of “Mastering OS-9” should label the disk in drive /d1 “Custom Master #2". Backup this disk.

That's all there is!

When all is done you should have a bootable disk with all your favorite features. lf your disk is unbootable or behaves erratically, edit your bootlist to reorganize the modules and try again.

Here are some tips on reorganizing bootlists:
* Put block-oriented device descriptors/drivers (such as ramdisks and disk drives) near RBF.
* Put sequential-oriented device descriptors/drivers near SCF.
* Place all the pipe modules together.

What you want to avoid is forcing OS-9 to map too much of the OS-9 system in a single 64K block. lf you have questions about this sort of memory allocation problem, read “The Dreaded BLOB” in the appendix.

You are not required to pipe the output of dsave to a Shell. You can further customize your new disk if you take the time to direct dsave’s output to a procedure file and then edit the file to exclude unneeded files or commands. Then execute the procedure file.

One last tip. Just because you have 512K does not mean that OS9Boot can be as large as you wish. Only include modules you need in a typical work session. For example, if you telecommunicate and like to capture information in a ramdisk buffer, install your terminal package’s modules and your ramdisk modules together, as we have done here. Or, suppose you run subLogic’s Flight Simulator II (tm) during your word processing work
sessions. lnclude Flight Simulator’s drivers with, say, the screen

driver for your particular word processor or spell-checker.

It is easy to see that you may want several boot disks, one for general purpose work and several others for specialized work. Personally, I use a general purpose boot then load additional modules as needed. But then I don't do a lot of gaming or programming on my system. I mainly use the Dynacalc spreadsheet and Bob van der Poel's DML9 mailing label/address database program for keeping FARNA System's books and the mail list for "the world of 68' micros" magazine, plus a few utilities to keep things in order. If I were doing any heavy duty programming, I'd probably make a boot especially for Basic09 and/or the C compiler. Good luck!

Time to open windows!

One reason to buy OS-9 Level 2 on the Color Computer 3 is that it offers true multi-tasking windows. Other computers are able to do this now, but it doesn't always work as expected.

Apple's System 7 for the Macintosh allocates memory in large chunks for at least four processes whenever MultiFinder (the multi-tasking module) is called. This wastes a lot of memory if all you want to do is switch between two programs. It also makes a 4MB machine rather slow... better have 8MB of RAM if you want to really use this feature!

Micro-Soft's Windows for the MS-DOS operating system handles memory a little better, but not multi-tasking. If a program was specifically designed correctly to run under Windows, then it will (should!!) multi-task correctly. MS-DOS programs don't always multi-task, sometimes causing the system to lock up because they expect to have complete control at all times. Sometimes Windows specific programs do this also, but not to often, and not if they were correctly programmed.

Level 2 compares favorably to OS/2 with Windows, costs less, and is a very stable system. Perhaps the only advantage OS/2-based systems have over OS-9 Level 2 is in the applications arena — but OS/2 hasn't exactly caught on over the years it has been out.

Logical devices

While you have only one physical terminal — your screen — OS-9 can create the illusion of having many more by creating "windows". In this book we create windows with the wcreate command. Each window can run its own Shell if you wish, making it act like another OS-9 computer on your desktop. You can also use a window to display program data output, as long

as the window is not running a Shell.

Recall that each device is recognized by OS-9 when the device's drivers and descriptors are in memory. The Level 2 disk comes with drivers and descriptors for seven windows named /w1 through /w7 plus a generic window descriptor named /w, all of which were placed in OS9Boot when we Configged the working system master in Tutorial 1. Confirming that they are in memory after boot is as easy as using the mdir command. As far as OS-9 is concerned, these windows might as well be physical terminals.

Creating windows -- text or graphics?

Each window "device" /w1 through /w7 comes with default sizes and colors. To establish a window with all its defaults you can use the iniz command. Not only is the iniz command somewhat beyond the range of "Mastering OS-9", but the defaults for many windows do not suit our purposes. For example, some windows were apparently defined for use by TV set bound CoCo 3's. No users of "Mastering OS-9" should be using a TV for a computer monitor.

With wcreate you can create a window to exactly match your needs despite a window's defaults. Window features which you control include:
* whether the window is for text only or for graphics
* the heighth and width of the window measured in terms of text characters
* the location of the window's upper left-hand corner
* the foreground, background, and border colors

Specifying each of these features requires some background knowledge.

First, what is the difference between a text-only window and a graphics window? Text-only windows, called hardware text windows, use hardware which quickly places text characters on your screen, making for extremely snappy screen response. However, you cannot draw circles, bars, lines and other graphics on text windows. Hardware text windows come in two types which we introduce in a moment.

Graphics windows can use a virtually unlimited number of fonts for text composition. They also allow you to draw and paint a wide variety of objects (circles, bars, arcs, etc.) intermixed with text. There is a drawback to this versatility, however, text

displaying is not quite as responsive as with hardware text windows. For many people, being able to use graphics makes slower text handling an acceptable trade-off.

chrarcters and pixels

The four types of graphics windows vary in resolution and number of colors available on-screen. We introduce these four types in a moment.

The two types of hardware text screens differ in the number of characters they can display. Type 1 displays up to 40 characters across; they can display no more because the characters are large. Type 2 displays up to 80 characters across with smaller characters. Both ordinarily display up to 24 rows of text.

Graphics screens come in four types differing in the size and number of "screen dots", or pixels, they display and in the number of colors each pixel can be. Each graphics window is ordinarily 192 pixels high. Depending on the type of graphics window you specify, the window may display up to either 320 or 640 pixels across. The number of colors available for each pixel varies from two to sixteen.

Table 7: window types and codes

There are 64 colors produced by the Color Computer 3. You can use up to any sixteen of them. See the Tandy-supplied Level 2 manual for information on how to choose specific colors.

Code	Displays
01	40-column text, up to 16 colors
02	80-column text, up to 16 colors
05	640 X 192 pixels, two colors
06	320 X 192 pixels, four colors
07	640 X 192 pixels, four colors
08	320 X 192 pixels, sixteen colors

What size window?

We'll only be using hardware text screens in "Mastering OS-9" but the concepts you learn in creating text windows are easily applied to creating graphics windows.

After deciding whether you want a text or graphics screen, the second question you must answer when designing a window is "How large will the window be?" Since window size is measured in terms of numbers of text characters displayed, another way to phrase the question is, "How many characters do I want to be able to display in the window?"

Wcreate measures window size in lines and columns of text.

Type 1 windows can display no more than 40 characters across, hence they are 40 columns wide. In contrast, a Type 2 window, 80 columns wide, can display 24 lines of text 40 columns wide and only use half your monitor screen.
If a large amount of text must appear on your screen, choose a Type 2 text window or high-resolution Type 5 or Type 7 graphics windows.

Where to put the window?

Once size is determined, you must specify the location of the upper left-hand corner. Do this by specifying the horizontal (or x) coordinate and the vertical (or y) coordinate. In this book, most windows are 80-columns wide. This means that the x-coordinate ranges from 0 to 79. The y-coordinate for all windows ranges from 0 to 23. The coordinates start in the upper left-hand corner at (0,0).

What color window?

The wcreate command requires you to specify foreground, background, and border color. Study this chart:

TABLE 7-2
Initial Type 2 window palette

0	WHITE
1	blue
2	black
3	green
4	red
5	yellow
6	magenta
7	cyan
8	white
9	blue
10	WHITE
11	blue
12	black

... and the process repeats, meaning that only the numbers 0 through 9 are defined.

Notice the way the colors cycle. This results from initial colors set up in the Color Computer palette for Type 2 windows. Since we only establish Type 2 windows in this book you can refer to this chart when creatlng wlndows.

If you choose another window type when using wcreate, the colors cycle differently. Sorry, but the logic to palette registers is beyond the scope of this book.

Ready to open a window

The syntax of wcreate includes choices from the issues discussed above: wcreate /wn -s = type xpos ypos width heighth foreground background border

Here's the procedure:
* Insert the window type you want by referring to the window type table above.
* Decide on the starting coordinates of the upper left-hand corner (the range of these coordinates depends on the type of window you choose) and insert these coordinates in the xpos and ypos places.
* Then, once again within the limits of the window type, select the width and heighth of your window in terms of lines and characters. Insert this data in the width and heighth locations.
* Use the color table above to choose codes for the colors you want — if you choose a Type 2 window. Refer to the Tandy manual for help on choosing colors for other window types. The border color refers to the screen space surrounding your main display. The background color is the "field" on which your letters are typed and graphics are drawn. Letters and graphics appear in the foreground color.

To open a Type 2 window occupying an entire screen with blue letters on a yellow background with a black border, type:

```
OS9: wcreate/w2-s=2  0  0  80  24  1  5  2
```

An example you've used before.

The number of the window you use is unimportant as long as it has not been previously initialized (that is, used as a parameter with iniz or wcreate or opened with display codes). Double-check the above command line. Will it do what we promise?

Here is the listing for make_1_2, the procedure file you used in the last tutorial when trying to fill the CoCo 3's memory:

```
echo creating 80 column text windows 1 and 2
*
* Creates two 80 column text windows using wcreate
* with blue letters on a white background
* and black letters on cyan.
* Notice that we only specify the border color on the
* first window created; the other window "inherits"
* that border color
```

```
create -z
/w1 -s=2  0  0  80  12  2  7  7
/w2  0  12  80  12  1  0
* wcreate with the z option demands a blank line after
* the list of windows is finished
* Here it is
shell i=/w1&
shell i=/w2&
*
*
Print message to user
*
echo "Press <clear> to select window screen"
echo "Aren't you glad you're Mastering OS-9?"
```

Apart from the comment lines (preceded by asterisks) and messages to be echoed, this procedure file is brief and simple. It uses wcreate's -z option which tells wcreate to expect the rest of its needed data from the standard input path. After the windows are created, "immortal" Shells are started in each (shell i=/w1&). These Shells are immortal in that won't "die" at the end of any work they do. It is not necessary to start a Shell in a new window. We show you a good use for a Shell-less window in Tutorial 10.

Closing windows

Wcreate initializes windows. lnitialization involves establishing a device in a device table, setting aside system memory for a path to the window, and other arcane system chores OS-9 handles in the blink of an eye. To remove a window, you must "de-initialize" it by name with the deiniz command.

Always de-initialize windows in an order reverse to which they were created. lf you create window /w2, then /w5, then /w7, you would have to de-initialize /w7 first, then /w5 and finally /w2.

Before you can de-initialize a window, you must kill any Shell running in it. You can kill a Shell with the "ex" Shell option. For example, type: **OS9: ex <ENTER>**
in the window you want to close. Then, assuming the window to be closed is window /w3, type from another Shell:

OS9: deiniz w3 <ENTER>

Moving on...

To summarize window-closing: If a window has a Shell, remove the Shell, then deiniz the window from another Shell.
If the window does not disappear try to de-initialize it again.

Concepts associated with wcreate are technically involved. Don't be intimidated. If the rules for wcreate are followed it is an

easy-to-use and helpful command. Oh — have some phone numbers handy!

Tutorial 7
Step 1: Booting up

TUTORIAL 7
STEP 1: BOOTING UP

Step 2: Creating windows

Find the backup of the disk created by OS9Gen in "Getting Started With Tutorial 7". Make sure it is labeled "Custom Disk #2 Backup". Place it in drive /d0 and boot up as usual.

STEP 2: CREATING WINDOWS.
2.1 Recall Make_1_2 from the introductory reading. Let's create a procedure file named Make_3_4 which creates two additional 80 column windows. We'll make decisions along the way concerning color and size.

OS9: load edit <ENTER>

We have two editing sessions back to back in this Tutorial. Loading edit saves some time. When we're done, we'll unlink it.

OS9:editmake_3_4 <ENTER>
E: -* <ENTER>

This makes sure we're at the top.

2.2 Type: **E: l*<ENTER>**
Nothing? Good. Just wanted to make sure you created a NEW file. It may happen one day that you will accidentally "create" a file with edit that already exists. Without warning the editor may let you change the file to a point where results are meaningless.

2.3 Now let's type in our file:

E: <SPACE> echo Creating 80 column text windows
3 and 4 <ENTER>
E: <SPACE> load wcreate <ENTER>
E: <SPACE> * <ENTER>
E: <SPACE> * Creates two Type 2 windows
<ENTER>
E: <SPACE> * with blue letters on white <ENTER>
E: <SPACE> * and black on cyan <ENTER>
E: <SPACE> * These are "vertical" <ENTER>

The last five lines above are comment lines. Two months from now, after you've written dozens of procedure files for window creation, you will appreciate these comments. Morale: ALWAYS comment your procedure files so you can easily folow what you did even years later!

2.4 Now we get down to entering the actual command:

E:<SPACE>wcreate -z <ENTER>

If you ever use the wcreate command from the "OS9: " prompt, you can just use the syntax:

OS9: wcreate /wn -s= type

followed by the location, size, and color parameters. But when using wcreate in a procedure file, it's convenient to use the -z option. Then the Shell looks ahead to the next few lines for the parameters it needs. It knows to stop sending these parameters to wcreate when it finally encounters a blank line.

2.4 So let's enter some parameters:

E:<SPACE> /w 3 -s = 2 0 0 40 24 2 7 7 <ENTER>

This is window /w3 and its parameters. This will be a Type 2 window. The next two numbers tell you that /w3 will have its top left-hand corner at the top left of your screen. This corner has coordinates (0,0). 40 and 24 signify 40 columns and 24 rows. Since this is a Type 2 (small font) screen, these 40 columns stretch only half way across your screen.

The last three numbers specify the foreground, background, and border colors. Whenever you define your first window in a series of windows, use -s = type as the first parameter; also make sure you define the screens border color. Use Table 7-2 for color references. lf you learn to set up your own palettes (not covered in this book) you should create a table like Table 7-2 for each.

Make sure you understand every parameter following wcreate above. lf you don't, read through the explanation again. Ater all, wcreate separates your computer from every computer in its price range — and then some. So get to be handy with wcreate!

2.5 Let's start another window:

E:<SPACE> /w4 40 0 40 24 1 0 <ENTER>

More obscure commands. Let's see, we're initializing window /w4 this time. But we don't use the -s=type parameter. Window /w4 will therefore inherit its type from the previous window definition. Also, we don't have to specify a border color as we want it to be inherited from /w3 also.

Where is the upper left-hand corner positioned? 41 columns across (the column labeled 40 is the 41st column because

programmers and mathematicians count starting with ZERO). This places the x-coordinate half way across your eighty-column screen. How far down do we start the window? The zero tells us to move down zero rows. So (40,0) is half-way across your-screen at the top. Proceeding along the list of parameters we see that the window has a size of 40 characters across and 24 lines down. It is the same size as window /w3. It uses color 1 for foreground and color 0 for background. With the current palette setting, that's blue letters on a white background.

2.6 Remember, when using the -z option with wcreate, the list of windows must end with a blank line. So we need to insert one:

E: <SPACE> <ENTER>

2.7 We want each of the newly created windows to have an immortal shell when the procedure file executes. This makes the windows ready to switch to and run commands:

E: <SPACE> shell i = /w3& <ENTER>
E: <SPACE> shell i = /w4& <ENTER>

2.8 Now for some additional comments and to print a message on screen:

E: <SPACE> * <ENTER>
E: <SPACE> * Print message to screen <ENTER>
E: <SPACE> * <ENTER>
E: <SPACE> echo "Press <CLEAR> to choose screen" <ENTER>

Why did we use quotation marks? Hint: Which symbols in that line are normally reserved for the Shell? If you already know the answer, pardon the pop quiz, move to the head of the class. If not, the answer is the "<" and ">", which are used to redirect Shell input and output. Since they are enclosed in quatations above, Shell knows they are just to be printed on screen as is. If they weren't in quotations, Shell would try to redirect I/O and return an error!

2.9 Since we won't be using wcreate again, we may as well get rid of it: **E: <SPACE> unlink wcreate <ENTER>**
When executed, this frees the memory wcreate took up. Wcreate is a fairly large command — 690 bytes (compare with setime at 280 bytes and makdir at 34 bytes) — yet it's still far smaller than the 8191 byte space it occupies in memory!

2.10 Let's move to the top of the file and check our work:

E: -* <ENTER>
E: l*<ENTER>

You should see the following:

```
echo Creating 80 colunn text windows 3 and 4
load wcreate
*
* Creates two Type 2 windows
* with blue letters on white
* and black on cyan
* These are "vertical"
wcreate -z
/3 -s=2 0 0 40 24 2 7 7
/w4 40 0 40 24 1 0

shell i=/w3&
shell i=/w4&
*
* Print message to user
echo "Press <clear> to choose screen"
unlink wcreate
```

lf all is well, go on to the next step. Otherwise use your editing skills to make your file look like the one above.

2.11 So now let's quit edit and execute our procedure file:

E: q <ENTER>
OS9: make_3_4 <ENTER>

Step 3: Editing startup

STEP 3: EDITING STARTUP
ON THE NEW SYSTEM MASTER.

3.1 First, let's see what's on the disk: **OS9:dir <ENTER>**

3.2 Go ahead and add either Make_1_2 or Make_3_4 to Startup. Take your pick. I suggest you only use one of the two procedure
files. Too many windows can get confusing

(Did you actuallv hear me complain about TOO MANY MULTI-TASKlNG WlNDOWS: No, I didn't think so!). Since Make_1_2 creates a pair of separate windows, it may be more useful. Of course, nothing is more useful than side-by-side screens for comparing hings (such as directories)!

OS9: edit startup <ENTER>
E: + * <ENTER>
E: <SPACE> make_1_2 <ENTER>
E:q <ENTER>
OS9:unlink edit <ENTER>

Step 4: Testing the new windows

Actually, since we're about to reboot, there is no need to unlink edit. But it's a good habit to get into.

STEP 4: TESTING THE NEW WINDOWS

Press the reset button on the back of your computer with the disk we just finished editing startup on still in drive /d0. Press the <CLEAR> button several times, checking to see that each window comes up as it should. lf all goes well, insert the disk marked "Phone List" in drive /d1 and continue adding names, addresses, phone numbers and occupations to your phone list. When done, don't forget to back up your work. Wouldn't want to have to start over from scratch if something happened to the master disk!

GETTING STARTED WITH TUTORIAL 8

Tutorial 8: Getting started The phone list

By now you've assemhled an impressive phone list. It's still not quite time to use it. Not until Tutorial 9 do you learn how to search for particular names and numbers, sort entries by last name, and find duplicate entries.

For now we ask you to continue to add to your list, so have more
phone numbers handy. Here are some suggestions:

* emergency numbers: hospital, ambulance, poison center
* baby-sitter numbers
* delivery pizza number
* public transportation schedule information number
* favorite vendor's numbers
* local dinner theater's phone number
* and so on.

What we'll cover Commands you will use -- **cobbler** *&* **dsave** *File fragmentation*

The **cobbler** and **dsave** commands, used together, can create a finely-tuned system master containing neatly stored files. Tutorial 8 shows you how easy this is to do.

While boot creation involves precision file placement, other files and directories may be located physically just about anywhere on a disk. In fact, a single file may exist in small parts all over a disk. This results when OS-9 cannot find contiguous free space on a disk to store a file. OS-9 then breaks the file into smaller pieces that will fit into the many smaller free spaces available. From a high-level perspective, the resulting file is logically intact while actually being physically fragmented.

The chief benefit of this approach to saving files is the optimization of the disk file storage. By spreading a file amongst unused spaces, you end up with less disk waste. The chief cost is that your drive heads must travel all over a disk to read a fragmented file. After weeks of creating and deleting files you may have a severely fragmented disk, resulting in slower system response when loading and saving files.

You can improve drive perormance by creating a new system disk in which all files are both logically and physically whole. This process requires two steps:

* First, create an OS9Boot file and Kernel on a new disk. Since these need to be carefully place, you should use a freshly formatted disk.
* Then, copy over all files from the old master disk to the new master. Each copied file will be logically and physically intact.

This second step relies on copy's ability to read a fragmented file and write it again physically intact when sufficient contiguous storage exists. Thus, given a freshly formatted disk in the destination drive, you can use copy to transfer a source file — fragmented or not— to an unfragmented destination file. As long as the destination disk's format is at least as roomy as the source disk's, no copied files will be fragmented.

ln this tutorial we examine cobbler as a painless solution to the first step above. For the second step we call on dsave, seen in action in Tutorial 7.

The benefits of **cobbler**

Cobbler provides an easy way to create a new boot which permanently reflects system patches. When you load OS-9 into your computer, dozens of modules are loaded into memory. These memory modules include data managers and device drivers and descriptors which allow your computer to talk to peripherals. Changes or improvements to your peripherals require new modules.

Well — not necessarily. Using modpatch and xmode often you can alter existing modules in memory to accomodate changes in hardware configurations. You have already seen modpatch at work as it changes the disk and keyboard drivers during Startup. Your Startup file also uses xmode to set your printer baud rate. Cobbler permanently saves these system changes by creating a new boot file with patched modules. Using cobbler is simple:

OS9: cobbler <device> <ENTER>

This command line creates a new OS9Boot and Kernel on the disk drive named in <device>. The disk in the drive should be freshly formatted.

Once you have cobblered a new disk you should create a CMDS directory on that disk and transfer GrfDrv and Shell into it to make it bootable. (Actually, GrfDrv is only required if you plan on using windows — which of course you do.) You probably will want to copy over other files, too.

Do NOT transfer the procedure files which streamlined the old system disk with xmode and modpatch. Since the new system disk automatically boots with a streamlined system, any attempt to modpatch the memory modules will only clutter your screen with an error message and slow down the boot proces. And since these streamlining procedure files are now irrelevant, you should remove their path names from Startup.

Copying files & directories with **dsave**

System disks can be stuffed with files and directories. Copying them by hand to a newly cobblered disk would be a mind-numbing task. Fortunately, **dsave** saves you the trouble.

Dsave creates a procedure file which includes a series of copy and makdir command lines to replicate the source directory's structure. The procedure file also includes appropriate chd commands to move around the source directory's subdirectories with no user involvement. Because dsave uses copy it can create an unfragmented version of a source disk when the destination disk is freshly formatted. In this regard, dsave is superior to backup. (It's also much slower than backup, however.)

Dsave can also copy data between disks with differing formats. If you have drives which allow more than 35-track single-sided operation, use dsave to transfer standard a Tandy-format disk's contents to a new larger-format disk

.The procedure file which dsave creates is executed by typing its name at the "OS9:" prompt. Before execution you can easily alter the file with edit. There are plenty of uses for an edited procedure file of this kind. We'll mention a few of them at the end of this discussion.

An example

Imagine you have a svstem disk in drive /d0 and a freshly formatted disk in drive /d1. Your goal i to create a new system master on the disk in drive /d1. lmagine further that your current boot-up process patched the drive step rate and the delay before key-repeat. It also set the printer baud rate to 9600 baud. These changes occur only in memory. The OS9Boot file on drive /d0 remains intact — and unpatched.

To create a new boot on drive /d1 based on the patched modules in memory, use the **cobbler** command:

OS9: cobbler /d1 <ENTER>

After cobbler finishes, a directory listing of drive /d1 would show
OSBoot as the only file. This new OSBoot reflects all changes made to your system since the most recent boot-up.

Planning your **dsave** *attack*

The next step in our thought-experiment is to use dsave. Plan your work hy answering these questions:0) Which directory do you want saved to the new disk? (Wwish to save the entire file contents of one disk to another disk; thus our answer is "The entire directory structure of the disk in drive /d0.")

1) Which device is the source device? That is, which device contains the directory we are saving? (In our case, this is drive /d0.)
2) Which directory is the destination directory? (For our purposes, the answer is, "The root directory of the disk in drive /d 1."
3) What name should we use for the procedure file which dsave will create (for now, use "Save.d0.d 1")?

Executing the attack

Once you know these answerst use Dsave in three steps.
1) First, change your current data directory to the directory determined in (0) above. In our case, we would type:

OS9: chd /dO <ENTER>

2) Next, execute the Dsave command with the following syntax (examine the syntax carefully):

dsave source.device destination.dir > file.name

This uses the answers to questions (1)-(3) ahove. In our imaginary example, we type:

OS9: dsave /d0 /d1 > Save.d0.d1 <ENTER>

Just for the sake of example, one possible output of the abovecommand line might be:

```
chd /d1
tmode .1 -pause
load copy
makdir CMDS
Chd CMDS
Copy /dO/CMDS/grfdrv grfdrv
Copy /dO/CMDS/shell shell
Chd ..
Makdir SYS
Chd SYS
Copy /dO/SYS/termset termset
Copy /dO/SYS/errmsg errmsg
Copy /dO/SYS/helpmsg helpmsg
Chd ..
Makdir COM
Chd COM
Copy /d0/COM/aif.cis aif.cis
Copy /d0/COM/aif.dlp aif.dlp
Copy /d0/COM/aif.wpr aif.wpr
Copy /d0/COM/icon.cis icon.cis
Copy /d0/COM/icon.delphi icon.delphi
Copy /d0/COM/F oicons.doc proicons.doc
Makdir PRO
Chd PRO
```

```
Copy /d0/COM/PROcis cis
Copy /d0/COM/PRO/nasabbs nasabbs
Copy /d0/COM/PRO/delphi delphi
Copy /d0/COM/PRO/ProStuff ProStuff
Chd ..
Chd ..
Copy /d0/ed ed
Copy /d0/pf pf
unlink copy
tmode .1 pause
```

The file Save.d0.d1 is created in the current data directory.

3) When all the work is done, execute the created procedure file by typing its name at the "OS9:" prompt. If the procedure file is not in the current data directory type its full path name. We type:

OS9: save.d0.d1 <ENTER>

Now the Shell executes this file as any other procedure file. It may contain dozens of lines of Shell comands, lines we would have had to type by hand if it were not for dsave. Dsave helped us reduce that work to three steps and only three command lines!

For each file copied with the imaginary procedure file Save.d0.d1, copy transfers a little over 4800 characters or bytes of information. The larger a file, the more transfers copy may have to make. This makes your disk drives work harder. The trick to better performance is to make copy's buffer as large as possible. You can increase the copy command's memory allocation with a memory modifier:

OS9: copy #24k this.path.name that.path.name

In this example, copy expands its buffer to 24k.

Dsave inserts a memory modifier in each copy command line it creates if you use this syntax:

dsave -s k source.device destination.dir > filename

where "k" is the buffer size in kilobytes used during copying. In our current example we would type:

OS9: dsave -24k /d0/d1 > Save.d0.d1 <ENTER>

in order to increase each copy's buffer to 24K.

Removing BoostStep and MudKeys

Without BoostStep and MudKeys on the new system disk, there is no use in calling them from the startup file. Consequently, after executing Save.d0.d 1 make sure you edit the new system disk's startup file to keep it from calling these two files. You

also use edit to remove the command line in which xmode boosts your printer's baud rate. Cobbler has already created a boot file reflecting the new baud.

Further uses for **dsave**

Fully describing the options of and uses for dsave can fill hundreds of pages. Examine the Tandy-supplied Level 2 manual for more ideas and then use your imagination. For example:

* You can create a procedure file for which produces a custom, unfragmented disk on another drive, or a procedure file for file archiving. Just edit the procedure file created by Dsave.
* Confused by the complicated directory structure on a disk? Use dsave's indentation option with the problem disk as the source device. The indentation option creates a procedure file which indents a subdirectory further to the right than its parent directory. Studying the file gives you a quick overview of the disk's contents and its structure.
* Use dsave to "backup" between disks with "incompatible" formats. For example, Tandy distributes OS-9 disks with 630 sectors on one side of a disk. Newer drives allow 1,440 sectors using both sides of a disk. Backup doesn't work unless both disks are identically formatted, soinstead use dsave to move the contents from a Tandy distribution master to a differently formatted disk.
* Use dsave's output to copy your database disk to a ramdisk. Edit out any files you don't absolutely need on the ramdisk. (Since dsave doesn't care about the physical format of the source or destination device, it's a natural choice.) Running a database off of a ramdisk is blindingly fast.

The options are limited only by the imagination.

Some final observations

Notice that a small change to a boot file or a procedure file affects the roles played by files on other parts of a disk. In this case, cobblering a new disk requires the removal of a few lines from your startup file. It also makes two procedure files, BoostStep and MudKeys, no longer necessary and candidates for deletion.

The consequences of a single change are sometimes considerable. If you do not anticipate these consequences you may see an error message or two crop up. Don't fret. Error messages, despite the scolding tone of their name, exists to help the user. With a little experience these error messages appear far less often.

TUTORIAL 8
STEP 1: SAVING YOUR
CUSTOM BOOT AND USING DSAVE.

1.0 Boot OS-9 with "Custom Disk #2".
1.1 Label a freshly-formatted disk with the name "Custom Disk #3" and place it in in drive /d1. Type:

OS9: cobbler /d1 <ENTER>

1.2 When drive /d1 stops whirring, type:

OS9: dir /d1 <ENTER>

1.3 Recall that when we used config to createyour current system master, we asked f or a f ull command set. Also recall that Shell has commands merged into it. There is no reason to duplicate in /d0/CMDS all the commands present in Shell.

OS9: free /d0 <ENTER>
OS9:del -x del <ENTER>

Notice we are deleting from the CMDS directory the command which deletes f iles. How can this be? Won't OS-9 need to have del in the CMDS directory in order to delete files? Actually, no. Del is one of the nineteen additional commands Microware merged with Shell. Even though Del will no longer exist in this disk's CMDS directory as a stand-alone command, it will always exist hidden in the command named Shell in the CMDS directory.

OS9: del -x link list load copy.old <ENTER>
OS9: del -x mergeprocsrename <ENTER>
OS9: del -x setime tmode unlink <ENTER>
OS9: del startup.old sys/mudkeys
sys/booststep <ENTER>
OS9: free /d0 <ENTER>

Now we have created a lot of additional disk space by getting rid of all the commands that are already merged with Shell. Unfortunately, we have also fragmented the disk!

1.4 We'll use dsave to copy all the files over to another disk. This will create an unfragmented copy of the original disk:

OS9: dsave -s16 /d0 /d1 > move.it.over <ENTER>

Dsave is now creating a procedure fil e named move.it.over in your current data directory which is the root directory of drive /d0. Wait patiently until the work is done. It won't take long.

1.5 Let's look at the procedure file dsave created:

OS9: list move.it.over <ENTER>

Notice two things. First, the letter “t” at the top of the procedure file. This forces the Shell to report to your screen each command line it executes. Second, tmode .1 -pause, also at the top of the file, ensures that the Shell won’t stop executing the file when your screen fills up. Press the spacebar if necessary to continue the listing. Notice that the last two lines of the procedure file help return your system to where it started.

1.6 Now let's execute the file:

OS9: move.it.over <ENTER>

Now THIS will take a long time. Go get a drink or a sandwich.

1.7 Let’s see what we accomplished:

OS9:free /d1 <ENTER>

You’ll notice the same amount of disk space is free except that now it i. all contiguous. That wasn’t so hard!

1.8 Well, since we won't need move.it.over again, let’s delete it:

OS9: del move.it.over

1.9 Now let’sremove linesfrom /d1/startup which call BoostStep and MudKeys, two procedure files whose patches are no longer required. We can also delete the xmode baud rate change line.

OS9: chd /d1 <ENTER>
OS9: edit startup <ENTER>

Use Edit surgery to remove the lines mentioned above.

Step 2: Testing the new disk

STEP 2: TESTING THE NEW DISK.

2.1 Remove “Custom Disk #2” from drive /d0. Put it aside. Put “Custom Disk #3” in drive /d0. Press the reset button on the back of your computer. This will cause OS-9 to re-boot.

2.2 When the boot process is complete, check your drive step rate, key repeat, and prionter baud rate. Use the <CLEAR> key to confirm the creation of your new windows. If any problems occur, check your work by reproducing this tutorial.

2.3 If there are no problems, find “Custom Disk #2”, remove the write-protect tab if there is one, and place it in drive /d1. Now let's copy “Custom Disk #3” over #2:

OS9: load backup <ENTER>
OS9: backup #56k <ENTER>

Follow the prompts.

2.4 Mark the disk in drive /d1 “Custom Disk #3 Backup” with the date. Mark the *other* disk “Custom Disk #3 Master” with the date. Make sure you do not press the disk with a pen or pencil point; write on a label, then apply the label to the disk.

STEP 3: PHONE LIST

3.1 When the backup process is complete, remove the disk from drive /d1.

3.2 Place the disk marked “Phone List and other files” in drive /d1.

3.3 Add more phone numbers to the end of the current Phone LF
file.

3.4 Quit the editor.

3.5 Back up your work.

3.6 Power down.

Getting started with Tutorial 9

GETTING STARTED WITH TUTORIAL 9

We finally put your phone list database to work. We also get to enjoy the handy utilities provided with the "Mastering OS-9" disk. Weprovide most of these specifically to manage your database. Utilitiesincluded on the "Mastering OS-9" disk are:

sort
grep
copy
convert
cls
count
uniq

All utilities except convert are written by Stephen B. Goldberg in OS-9 Macro Assembly language. Convert is written in Basic09. Still, convert has another Goldberg utility merged into it. There is also a "BONUS" directory with a few public domain utilities that have been mentioned throughout this book.

Utilities written in Macro Assembly language are extremely fast and take up almost no memory space. Basic09, too, is often fast enough for utilities. Dale Puckett's "The Official Basic09 Tour Guide" recounts some BASlC benchmarks which show that BasicO9 is amazingly fast, perhaps still the fastest and most elegant BASlC on any microcomputer. (See Dale Puckett's article, "Start Basic09", at the end of this book.)

In Tutorial 6 you merged these utilities together and stored them in your backup system master's CMDS directory under the name copy. Then youplaced a line in your Startup file which loaded Copy into memory. Now that they are ready for us in memory, let's get ready for them by learning a few tricks on how to tap their power.

Path redirection

The three input/output redirection techniques described below make it possible for you to create extremely powerful comand lines. Just assemble a series of simple commands which process your data to do a larger job. This power is especially enhanced when using the Goldberg utilities.

Input redirection

To recap what we know of I/0 redirection:

* lnput TO a command often comes from your computer keyboard. lf input comes instead from a disk file or from another terminal, you have input redirection. To redirect input to a command from a file you type:

OS9: command.name < file.name <ENTER>

where the "<" tells command.name to get its input from file.name and not from the standard input path (your keyboard).

Example: Sort acts on files by input redirection. Use:

OS9: sort < phone lf <ENTER>

to sort your phone list. lf you need more memory try the #K memory modifler. For example,

OS9: sort#32k < phone lf <ENTER>

In both cases, the output goes to your screen.

Example: The count utility counts the number of characters, words, and lines it receives from the standard input path. If you want to know how many characters, words, and lines are in your phone list type:

OS9: count < phone lf <ENTER>

If you just want to know the number of lines type

OS9: count -l < phone lf <ENTER>

The number of lines in your file is probably the number of phone numbers you entered. The command line above therefore informs you about paper needs when listing Phone LF to your printer.

Output redirection

* Output FROM a command can be redirected to a disk file or to your printer.

Example:

OS9: dsave /d0 /d1 > file.name <ENTER>
OS9: list file.name > /p <ENTER>

In both examples if you had left off "file.name", the output would have gone to the standard output path (your screen).

Example: You'll need to sort your database from time to time. When you do, sort accepts Phone LF as its input and outputs a sorted file. Redirect that output to a temporary file named, say, TEMP: **OS9:sort < phone lf > temp <ENTER>**

Then you can delete the original (unsorted) phone list and give TEMP the name Phone LF.

OS9:delphone lf <ENTER>
OS9:rename temp Phone LF <ENTER>

Pipes: a third way to redirect

* A command can also receive data from — and give data to — another command. The data sent from one command to another is put temporarily in a memory buffer called a pipe. The first data into this buffer is the first data out of it.

Example: **OS9: d !count -1 <ENTER>**
This command line returns the number of files and directories in your current data directory. How does it work? The output of D is a directory listing with one file or directory name per line. Count with the -l option counts the number of lines lt receives from the input path and prints that number to your screen. You can use Count to count the number of phone numbers in Phone LF — assuming you have one name and phone number per line.

OS9: count -1 < phone lf <ENTER>

grep - *the one you'll wear out*

Of all the utilities provided you'll use grep most often. Grep searches through a file (call it Target.file) for a string of characters which is your target text.

OS9: grep target.text target.file <ENTER>

If you wanted to find Tom Smith's phone number, you might try:

OS9: grep Smith phone lf <ENTER>

Grep would print each line which contains the string "smith":

Smithsonian, Air & Space Museum\ 555-1212
Smith Brothers Jewelry and Auto Repair\555-1212
Smith, Tom\911-9090
Jackson, Tom\goldsmith\555-1212
Aerosmith Fanclub Of Outer Loop\555-1212

Notice in this example that Grep is not case sensitive. Also, the string may be part of another string. Substrings such as "smith" are usually not a problem; imagine, though, if the substring were "the" or "ing". Even though you may be looking for Erin the Great's phone number, or Ingot International, be prepared when Grep returns pages of possible
phone numbers. Be simple enough in choosing a target string to ensure you find your number; be speciflc enough to ensure you don't get a hundred numbers listed to your screen.

Grep with pipes

If you have a large directory (LARGE. Dl R) which may or may not contain a file you need (f ilename), try these command lines to track it down:

OS9: chd large.dir <ENTER>
OS9: d ! grep filename <ENTER>

D outputs a line-by-line directory listing. lf the command you seek is in that directory, Grep will find it in the list which D sends it through the pipe. When the filename is found, Grep prints the filename to the screen. Or, if you need to know the

hex size of a file, you can type

OS9: dir e ! grep filename <ENTER>

Dir e will send an extended directory listing to the output path. Grepwill print any line containingfilename, so all thejuicy data associated with filename will be printed to your screen — including its size in hex notation.

Space does not permit us to experiment with all the features of the Goldberg Utilities. You can find complete explanations of their syntax in the Appendices. Further examples and complete documentation for each and all of Stephen Goldberg's utilities are separately available. There are dozens of other utilities you may find interesting. If you find the provided utilities useful (which they are — why else would they call them utilities?) you can purchase more of them through FARNA Systems and other dealers. Call around for details.

This is a short tutorial. lt's designed to be fun. lt also contains a surprise so pay close attention at the beginning. The end of the tutorial is free-form. And no, you don't have to add any more phone numbers to the list if you don't want to!

Tutorial Nine
Step 1: Using **count**

TUTORIAL NINE
STEP 1: USING COUNT

1.1 Boot OS-9 with "Custom Disk #3 Backup".

1.2 Place "Phone List and other files Backup" in drive /d1. Type the following:

OS9: chd /d1 <ENTER>
OS9: dir <ENTER>

Do you see Phone LF? lf not, you have probably placed it in a directory other than the root directory. Use chd to move to that directory if necessary.

1.3 Type: **OS9: mdir <ENTER>**
Make sure **grep, sort, count** and the other utilities are in memory. They should be since you loaded copy during startup.

1.4 Now let's display our phone list:

OS9: tmode . 1 pause <ENTER>
OS9: list phone_lf <ENTER>

1.5 Quite a listyou have,there. How many phone numbers is that? We'll find out by using **count**:

OS9: count -1 < phone_lf <ENTER>

1.6 To f ind out how much memory you'll need to sort this file, count the number of characters in it.

OS9: count -c < phone_lf <ENTER>

Step 2: Using **grep** *and* **sort**

STEP 2: USING GREP AND SORT

2.1 Think of a name on your list. Say the name is "Ward ". Now let's try to find that name in our list:

OS9: grep w phone_lf <ENTER>
OS9: grep wa phone_lf <ENTER>
OS9: grep war phone_lf <ENTER>

Notice how the list narrows after each line is entered. You probably saw the name you were searching for after the first line. You can quickly find any name in your list just by looking for a couple characters in the name.

2.2 **OS9: grep 23 phone_lf <ENTER>**

You see a list of people whose phone numbers include the string “23”.

2.3 Let’s see the numbers in alphabetical order:

OS9:grep 23 phone_lf ! sort <ENTER>

2.4 How many phone numbers contain the string “23 ”? Don’t care to count them yourself ? Try this:

OS9: grep 23 phone_lf ! count <ENTER>

2.5 Suppose you had to call all your friends in a long-distance exchange such as Washington, D. C. No need to play favorites so alphabetize them:

OS9: grep 202 phone_lf ! sort <ENTER>

2.6 If you included addresses in your phone list grep can come in even handier. Going on a trip to Virginia? Find out how many people you know there:

OS9:grep VA phone_lf ! count <ENTER>

2.7 Choose a generous sort buff er. For example, if your file is 7,000 characters, try 16K or more. In case Phone_LF is really large, the following example uses a 32K buffer. Sort handles larger buffers too, up to 56K.

OS9: sort #32k < phone If > temp <ENTER>

How long do you think this will take? Surprised? Imagine sorting
these by hand!

2.8 Now let’s clean up the old file:

OS9: rename phone_lf old.phone <ENTER>
OS9: rename temp phone_lf <ENTER>
OS9: list phone_lf <ENTER>

Looks neater!

2.9 Suppose you put your acquaintance’s occupations in with the name and phone number like so:

Jones, Tom\Singer\203-555-1212
Jones, James Earl\Actor\213-555-1212
Welby, Marcus\Doctor\601-555-1212-

You can create a seprate file of doctors easily:

OS9: grep doctor phone lf ! sort > Doctor.file <ENTER>

Step 3: Check for duplicates: **uniq**

STEP 3: CHECKING FOR DUPLICATES

3.1 The **uniq** command eliminates consecutive redundant lines. If you have entered the same line of text twice, this command sequence will catch the error and correct it (Phone_LF needs to be sorted immediately before using uniq):

OS9: uniq < phone_lf > temp <ENTER>
OS9: del phone_lf <ENTER>
OS9: rename temp phone_lf <ENTER>

Step 4: The surprise: a **ramdisk**

STEP 4: NOW FOR THE SURPRISE

4.1 When OS9Gen ran in Tutorial 7 it created an OS9Boot file with a ramdisk built in. Check memory to f ind r0, the ramdisk descriptor:

OS9: mdir <ENTER>

4.2. Iniz and format /r0:

OS9: iniz /r0; format /r0 <ENTER>

Follow the prompts.

4.3 Now let's check our new drive:

OS9: dir /r0 <ENTER>
OS9: free /r0 <ENTER>

4.4 We're going to use dsave to generate a procedure file to move the contents of /d0/CMDS to /r0. Then we're going to use edit to change the procedure file to copy over only those files and commands we need.

OS9:chd /d0/cmds <ENTER>
OS9: dsave -s24 /d0 /r0/cmds > /d0/Move.to.ram <ENTER>

4.5 When dsave finishes, we'll edit the file Move.to.ram:

OS9: edit move.to.ram <ENTER>

4.6 Now, at the "E:" prompt, delete the lines which copy commands you don't normally need. Next, add a line at the beginning of the the file to create /r0/CMDS with the makdir command. Last, add a line at the end which copies Phone_LF to the ramdisk.

E: <SPACE> makdir /r0/CMDS <ENTER>

E: <SPACE> copy#32k /dl/phone_lf phone_lf <ENTER>
Quit the editor.

4.7 Assuming you've checked your work carefully, let's execute this procedure file:

OS9: chd /d0; Move.to.ram <ENTER>

4.8 When the work is done, change your current execution and data directories:

OS9: chd /r0; chx /r0/cmds <ENTER>

4.9 Now tryGrep withyour phone list on the ramdisk:

OS9: grep bow phone_lf <ENTER>
OS9: grep h-L phone_lf <ENTER>
Amazing how fast it works!

Step 5: Some quirks with ramdisks and with copy

STEP 5: SOME QUIRKS WITH RAMDISKS AND WITH COPY

5.1 Ramdisks and their data are lost when you power down. If, at the end of a work session, information in the ram disk is more to your liking than the original inf ormation on the floppy disk drive, you'll have to copy the files from the ramdisk back to the floppy. In this case, you might type:

OS9: copy /r0/phone_lf /dl/phone_lf <ENTER>
This assumes your data directory was changed to the floppy you want to copy the file to. Then type "y" when the "Rewrite :" prompt appears.

5.2 An even easier approach would be to enter the editor to create a procedure file to do the same thing.

OS9: edit backup.ramdisk <ENTER>
E: <SPACE> echo " * * * Copying phone list from /r0 * * *" <ENTER>
E: <SPACE> echo " * * * back to drive /d1 * * * " <ENTER>
E: <SPACE> copy #32k /r0/phone_lf /d1/phone_lf <ENTER>
E: <SPACE> y <ENTER>
E: <SPACE> echo " * * * all done! * * * "
Stephen Goldberg's version of the Copy command ask.s if you want to rewrite an existing file. Since many of the files on drive /r0 exist on drive/d0, you must insert "y" after a line which requests rewrite permission. At this point, quit the editor.

5.3 Test the new procedure file:

OS9: backup.ramdisk <ENTER>
Please experiment with the Goldberg utilities and all other Level 2 tools. You'll f ind it more fun than you ever imagined!

GETTING STARTED WITH TUTORIAL TEN

Getting started with Tutorial Ten

Most people who buy computers start out using them for one major application — just word-processing, just spread-sheeting, or just games. At this point in their computer experience, multi-tasking windows are more than they need.

When a computer user expands his or her software library, they reach a bottle-neck since most computers only allow the execution of one application at a time. This is true despite the gallant efforts of top programmers.

Level 2 with windows on the CoCo 3 was designed from the start to avoid barriers MS-DOS computer systems trip over. Level 2 windows provide convenient access to many appli-catlons at once. Level 2 thus provides a powerful solution to computer users tired of ordinary PCs with ersatz windowing and patchwork multitasking. Even though MicroSoft Windows does multi-task, it doesn't do so as smoothly or in the same way as OS-9 systems.

Using startup to streamline your wor

After acquiring several applications comes the task of starting each in its own window. This job can be delegated to the startup file. In Tutorial 10, we show you how.

Another consequence of multi-tasking on a floppy disk-based system is disk-swapping. Users who by habit put data disks in drive /d1 must constantly switch disks in and out of that drive when saving output from the several programs running on the CoCo 3.

A solution to this is to use the ramdisk provided with “Mastering OS-9” as a “data disk”Not only is a ramdisk convenient in a disk-intensive operating system such as OS-9, it is fast. Just make sure you copy data files from the ramdisk to a floppy often enough to avoid severe depression in the event of a power failure or you turn the computer off (no pwer and all data from the ramdisk will be lost!).

Redirection saves the day

ln Tutorial 7 you learned that Wcreate creates a window of a certain size with certain user-selectable colors. After creating such a window you may start a Shell in it. From then on it acts as another OS-9 Color Computer, ready to accept your com-mands from the standard input path and ready to send output and error messages to the standard output path.

A program other than Shell can take control of that window when you open the standard data paths between the window and the application. For example, if you want the word-processor WordKing to run in its own window you first create a window without a Shell.

OS9: iniz /w7 <ENTER>

Then type the following command line:

OS9: wk < > > /w7& <ENTER>

The hieroglyphics between wk and /w7 actually make sense. The first symbol (<) you probably recognize as the redirection modifier associated with the standard input path. The following symbols redirect the standard output and error paths. Thus wk receives its input when you select /w7 with the <CLEAR> key and start typing. Wk also sends its usual output and all errors to / w7 for your viewing. The ampersand (&) forces wk to work as a background (or f orked) process. The Shell which executes the above command line is then free to continue other work.

ln general, you can insert a command line similar to the one above into your startup file (or into a procedure file called by startup). Just replace wk with the name of the application you wish to start automatically at boot.

We've mentioned a couple of times that a window created to display an application through path redirection (as in the command sequence above) should not contain a running Shell.

Consider what would happen if Window /w7 had a Shell running in it when you executed the above command line. For one thing, visual chaos would ensue as each program competes in outputting information to your screen. Actions taken by your commands would lead to confusing and perhaps disastrous results. The basic problem is that the auto-executed application AND the Shell are competing with each other for I/0. Be sure not to redirect paths to a window which has a Shell already present.

Automating the ramdisk

During Tutorial 9 you manually inized and formatted the "Mastering OS-9" ramdisk. In this tutorial we create a procedure file to automate the same job. Then we will edit startup to call this file automatically. The ramdisk will then be ready for use right after booting.

We will also write a procedure file to copy your phone list to the ramdisk. After hootup, phone numbers are seconds away from your fingertips.

lncidentally, the ramdisk provided is large. You can alter the size
of it with Kevln Darling's dmode utility, also included on the "Mastering OS-9" disk. Darling's philosophy about ramdisks: "They really shouldn't be large because they should only be used for temporary storage." Use a large ramdisk and you risk losing a lot of data. Be cautious! Make copies of ramdisk files fairly often. You won't need to copy your OS-9 commands back to floppy, of course.

This is personal computing!

OS-9 Level 2 gives you tools to get your work done quickly and professionally. With a little preparation and the ideas offered in "Mastering OS-9", you are on your way to a work style that can only inspire admiration. Personal computing standards are higher than ever before and Level 2 is here to meet them.

Tutorial 10
Step 1: Putting Phone List on /r0

TUTORIAL 10
STEP 1: PUTTING THE PHONE LIST ON RAM DRIVE /R0.

1.1 Remove any write-protect tab on "Custom Disk #3 Backup". Boot OS-9 with that disk.

1.2 Place "Phone List and other file Backup" in drive /d1. Type the following lines:

```
OS9: chd sys <ENTER>
OS9: edit move.phone <ENTER>
E: <SPACE> * Moves phone list to ram disk
   <ENTER>
E: <SPACE> * Make sure ram disk is inized and
   <ENTER>
E: <SPACE> * formatted first <ENTER>
E: <SPACE> copy /d l/phone lf /rO/phone
   <ENTER>
E: <SPACE> echo "Phone list in ram"
   <ENTER>
E:q <ENTER>
```

1.3 The above sequence creates the procedure file "move.phone". The comment lines spell out what it does.

Step 2: Formatting the ramdisk

STEP 2: FORMATTING THE RAMDISK.

2.1 Type the following lines:

OS9: edit iniz.ram <ENTER>
E: <SPACE> t
E: <SPACE> * initialize r0 <ENTER>
E: <SPACE> iniz r0 <ENTER>
E: <SPACE> * format the ramdisk <ENTER>
E: <SPACE> * NEVER execute this proc file <ENTER>
E: <SPACE> * if /r0 is not empty <ENTER>
E: <SPACE> * Otherwise, bye-bye data <ENTER>
E: <SPACE> format /r0 r "DarlingDisk" <ENTER>
E: <SPACE> -t
E: q <ENTER>

2.2 Now we have a procedure file that initializes and formats our ramdrive.

Step 3: Starting Apps with Path Redirection

STEP 3: STARTING APPLICATIONS THROUGH PATH REDIRECTION

3.1 Type the following lines:

OS9:edit app.start.5 <ENTER>
E: <SPACE> * Create window 5 <ENTER>
E: <SPACE> wcreate /w5 -s=2 0 0 80 24 1 0 0 <ENTER>
E: <SPACE> echo Starting application in w5 <ENTER>
E: <SPACE> edit #32k < > > > /w5& <ENTER>
E: q <ENTER>

3.2 We now have a procedure file that creates an 80 column by 24 line window. The next to last line starts edit in this window. When "app.start5" is run, a window will be opened and edit will be started and ready to run.

When you buy an application which you want to run on startup, put its name in place of edit's in the last line. You may also have to resize the window using wcreate.

Step 4: Putting it all together

STEP 4: PUTTING IT ALL TOGETHER.

4.1 Type: **OS9: copy startup startup.old <ENTER>**
Go ahead and rewrite Startup.old if copy asks.

4.2 Now let's change startup:

OS9:edit startup <ENTER>
E: + * <ENTER>
E: - <ENTER>

Repeat until you place the edit pointer just after the current last line in the file.

4.3 Once the pointer is properly placed, we can begin editing the startup file:

E: <SPACE> sys/iniz.ram <ENTER>
E: <SPACE> sys/move.phone <ENTER>
E: <SPACE> sys/app.start.5 <ENTER>
E: - * <ENTER>

4.4 Check your work. Make any corrections necessary. Ensure that iniz.ram appears in Startup bef ore any procedure f ile which
relies on the ramdisk. Do you remember how simple the Startup file used to be?

4.5 If the file checks out okay, we can quit the editor:

E: q <ENTER>

4.6 Test the disk by pressing the reset button to reboot. If all works well, why not, just f or f lash, List your phone list to each of the two side-by-side windows you included in your boot f ile. Watch the list scroll simultaneously in both windows! You can't do that on just any computer...

Wh ere To Go For M ore lnformation

Although Tandy has sold millions of Color Computers, support for the Color Computer is surprisingly small in scope. Small, yes, but mighty. There are still several user groups ranging from small local organizations to nationwide services.

There is only one magazine in regular print now, and that is "the world of 68' micros". This magazine has been in existance for about two years. It supports Disk Basic, and OS-9 on the Color computer as well as OS-9/68000.

Books on OS-9 are rare. The excellent Tandy manual is required reading. Since it is designed not as a tutorial but as a reference manual, do not expect to understand everything. Keep coming back to it as your OS-9 skills improve. You'll get more and more out of it each time.

The same is true of Kevin Darling's book, "Inside OS-9", once available from FHL (its publisher). The more I come back to it, the more I enjoy it. Although Kevin's book is filled with charts full of "hacker" information, he also takes the time to explain in lay terms some confusing aspects of Level 2 and the CoCo 3's hardware. The book's enthusiasm is infectious, too. It's obvious Kevin enjoys what he does.

Dale Puckett and Peter Dibble have written two books together on OS-9. "The Rainbow Guide to OS-9" and "The Complete Rainbow Guide to OS-9 Level lI", published by Falsoft, Inc., are still available from the publisher. The first provides an overview of Level One OS-9 (the predecessor to Level 2 on the CoCos 1 and 2) but also provides general information about Level 2 and about the OS-9 Ethos — how it works and why it works. It is safe to say that the majority of CoCo OS-9 users learned the ropes from these books.

The second book by Puckett and Dibble is oriented more for the beginning user, as it takes you on a casual, hands-on guide to windows. Readers have found "The Complete Rainbow Guide" the best book by these two authors yet. It covers windows and BasicO9 programming under Level 2.

For information on telecommunicating, you can't beat Alfred Glossbrenner's "The Complete Handbook of Personal Computer Communications" (St. Martin's, 1985). It is a fountain of useful and entertaining information. lt includes information on Delphi, one of the newer information service whose most active

participants are arguably CoCoists. lt offers ease-of-use, low prices, and a wide range of databases and forums, including Dow Jones and online shopping. Danny Goodman's "DELPHI: The Official Guide" is required reading.

lf you can get to a CoCoFest, do it. Take along some cash and be prepared for an onslaught of fine products. I have never met anyone who regretted going to a CoCoFest. There are currently two held annually. The largest is held in Elgin, Chicago near the end of April/first of May. Contact the Glenside CoCo Club for information. The Atlanta Computer Society sponsors the Atlanta CoCoFest each October.

Above all else, join a CoCo Club — or start one yourself. One of the most surprising facts about the Color Computer is that so many were sold and so few CoCoists know other CoCoists! The chances are good that other CoCo owners live just around the corner — and one of them might just be adept at OS-9!

Hardware for Your CoCo 3/ OS-9 Level 2 System

(c) Sept. 1988 by Marty Goodman
Updated by F.G. Swygert, 1995

OS-9 is uniquely suited to easy adaptation to a wide variety of CoCo hardware set-ups. Once proper device drivers and device d escrlptors have been devised for a given hardware item it can often be used with any existing OS-9 application. n this respect, OS-9 is considerably superior to the primitive operating system erringly referred to as “RS-DOS” (the built-in Disk Basic CoCos come with). This is not really an operating system at all, just a set of patches to BASIC that allow access to a disk drive.
Let’s look at a variety of hardware accessories and means of adding them.

Floppy Disk Controllers

The one essential add-on card for running OS-9 is a Floppy Disk Controller, which enables your computer to be attached to disk drives. There are a great number of different Floppy Disk Controller (FDC) cartridges available. The Tandy FD502 cartridge combined with a double sided disk drive was originally designed for the CoCo 3. All previous Tandy controllers EXCEPT their earliest (Cat No. 26-3022) will work fine with a CoCo 3 OS-9 system and any standard 5.25" or 3.5" disk drive. You CANNOT use the Cat No. 26-3022 disk drive cartridge with a CoCo 3. This cartridge requires 12 volts which is not supplied from the CoCo2 or 3 cartridge port. While an MPI will supply the necessary 12 volts, the cartridge will not tolerate running at high speed. OS-9 Level 2 runs at double speed all the time. This cannot be disabled or the cartridge easily modified.

A lot of other companies have made disk controllers for the CoCo. Hard Drive Specialists and J&M Systems have delivered thousands of economical and reliable disk controllers. CRC / Disto made a Super Controller that functions perfectly well as an FDC but also provides a minibus, described below. CRC/Disto also made a copy of the Tandy controller, the Disto Mini Controller.

No Halt Controllers

The original design for CoCo disk controllers allowed the disk controller to completely halt the operation of the 6809 during parts of the operation of the disk controller. Under Disk Basic this wasted a little time but was never much of a problem.

However, under OS-9 this became unpleasant and, for many, unacceptable. It meant that during floppy disk access no other processes could operate. Characters being typed at the keyboard would be dropped, data arriving at the serial port(s) would be missed and so on. This problem was addressed by the makers of sophisticated “No Halt” controllers. Such controllers do NOT interrupt the operation of the 6809, and so provide considerably faster, cleaner operation under OS-9.

Note that these controllers typically offer no special advantage under Disk Basic. They operate just like the older controllers.

Three companies have made No Halt floppy disk controllers: Sardis (the first to do so), Performance Peripherals, and CRC / DISTO. Performance Peripheral’s boasts low power consumption and exceedingly fast operation. The Disto controller is also low power, and boasts support for the Disto Mini Bus array of add-on cards.

Adding Expansion Capability to your CoCo 3
MultiPak Interface:

The MultiPak is a device that plugs into your Color Computer’s 40 pin expansion port. It adapts the CoCo to accept up to four different plug-in cards.

Internally, the MultiPak provides “buffering” for the fragile address and data lines coming out of the CoCo 3. It also provides added power allowing extra cards to run on the CoCo 3. To use most add-on interface cards with a CoCo, a MultiPak Interface is required. This especially includes most brands of hard disk drive adaptors and many varieties of serial and parallel ports and real time clocks.

Catalog number 26-3024 type MultiPaks will need to be “upgraded” by the addition of a new CoCo 3-compatible PAL chip, available from Small Graf-X Etc. Catalog Number 26-3124 type MultiPaks require the addition of a satellite board, which can be done by Tandy repair or by referring to a schematic in the October 1988 issue of Rainbow Magazine or FARNA Systems' excellent CoCo reference book, “Tandy's Little Wonder” .

Catalog Number 26-3124A MultiPaks do not require any upgrade in order to work with the CoCo 3. Pricing on the MultiPak will likely range widely, from $20 for an older model if you can find one, to as much as $70 for the 26-3126A model.

The MultiPak Interface hardware, under software control, switches around the *CART interrupt line from slot to slot. Some sophisticated OS-9 hackers may want to defeat that slot select hardware and simply "hard wire" all *CART interrupts from all four slots direct to the *CART line. This may prevent occasional subtle problems with some OS-9 hardware.

Y Cable:
Some folks try to "economize" on money or space by using a Y cable instead of a MultiPak Interface. I personally do NOT recommend Y cables because they have on occasion been associated with causing unreliable disk I/0. The CoCo was never designed to run its system bus over ribbon cable. However, some folks are able to use them, especially if the Y cable is short .

Disto Super Controller Mini-Bus:
CRC/Disto sold two types of Disk Controllers and one ramdisk board that supports a special mini-bus onto which one can plug one of several Disto / CRC add-on cards. Such cards can provide support for hardware RS232 ports, parallel ports, real time clocks, and/or Hard Disk adaptors.

What is especially important is that, if you use a Disto/CRC controller with the custom Disto/CRC minibus cards, you will not have to purchase a MultiPak.

Add On Options
RS-232 hardwareports:
Although the CoCo 3 comes with a kind of "serial RS-232 port" on the main box, this port is almost useless under OS-9. The reason is that the CoCo 3 has no hardware to handle conversion of parallel to RS-232 serial data, a conversion which requires careful timing. Instead, such conversion must be done by the 6809 inside the CoCo, eating up an unacceptable amount of processor time under OS-9.

The main application for that "bit banger" serial port under OS-9 is driving a serial printer. The CoCo basically only communicates with a printer "one way", so the load on the processor is not as great as in true "duplex" communication. For other serial communications, particularly telecommunications via modem,you must acquire an add-on hardware UART-based RS-232 port.

The original means of adding such a port was that of plugging a "Deluxe RS232 Pak" (cat no. 26-3226) into your MultiPak. Years ago, PBJ Corp. offerred their "2SP Pak" that provided two hardware serial ports. CRC/Disto offered a plug-in card that works with a MultiPak and functions about the same as the old Tandy "Deluxe RS-232 Pak". Disto also offers serial port capability in the form of an option on one or more of their custom minibus cards. CoCoPro! once sold a converted Modem Pak that worked as a basic RS-232 port. Plans for converting a Modem-Pak are still available.

The only RS-232 port currently in production is made by CoNect. They have a standard port, a double port, and a special high-speed serial port (also available as a double port).

Parallel Ports:
Many printers come with only a parallel port. Color Computer users have grown accustomed over the years to purchasing a serial to parallel converter in order to use their paralel-only printer with the "bit banger" (4 pin DlN) serial port on the main Color Computer. Under OS-9, however, printer output can easily be re-directed to any hardware card. Many OS-9 users have taken advantage of this to install a parallel port on their computer.

J&M systems made a disk controller that had a parallel port built in. Disto/CRC made an add-on parallel card and several models of mini-bus cards that have a parallel port.

Real Time Clocks:
One of the desireable features of OS-9 is the fact that files are marked with the time of their latest update. But this also means that the system needs a means of knowing what time it is. It is rather a nuisance to enter the date and time at each boot-up.

Real time clocks solve this problem. They use an ultra low power clock, typically powered by a long life lithium battery. Once set, such a real time clock remembers the time even when the main power is off. Via software, that clock can be programmed to supply OS-9 automatically with the correct minute, hour, day, and date every time you boot up.

Disto/CRC made several mini bus cards that have a real-time clock. Spectro Systems supplies a Dallas Semiconductor "SmartWatch". OS-9 drivers are available for down-loading on

Delphi or from FARNA Systems on their “Patch OS-9” disk.

Floppy Drives (the physical drive):

The original Color Computer came with a 156K 35 track 5.25" single sided disk drive. Over the years, 40 track single, then double sided disk drives became standard for OS-9 systems.

Then OS-9 users began to switch to the higher density 0 track double sided 5.25" drives. Many have since converted to using the now increasingly standard 3.5" 80 track double sided drives. These 720K drives do come in handy with OS-9, especially if one doesn't have a hard drive. Note that the older, now rare 5.25" 720K drives are electrically identical to the 720K 3.5" drives. The difference is that the 5.25" modles can be “double-stepped” to read standard 360K double sided and 156K single sided 5.25" disks. Note also that the newest 1.44MB drives will work as 720K drives. “Real” 720K 3.5" drives are getting more difficult to find every day.

All of these drives can be controlled by the SAME disk drive controller. I recommend, however, keeping in your system at least one 40 track double sided 5.25" drive, to be able to both read from and write to disks made by other OS-9 users.

ln general, the 1.2 megabyte 5.25" drives and the 1/4 megabyte 3.5" drives cannot be used on the CoCo (though one VERY “high end” system was engineered by Hemphill Electronics that actually supports such drives on the CoCo.)

Fans:

While a properly designed linear power supply of the sort used by the Color Computer should NOT need a fan, but should rather cool adequately by convection, some owners of CoCo 3’s do find that their machines overheat.

The easiest fix for this is adding a fan. Typically a low power AC brushless motor fan works fine. These are often available at electronic surplus stores for $5 or less each.

Hard core hardware tinkerers will, instead, want to consider removing the less than adequate power transistor Tandy uses on the 5 volt supply of the CoCo 2, and instead hook up a 2N3055 mounted on a massive, finned heat sink. This should cure any tendency of the power supply to give out due to overheating, unless the problem is in the transformer or capacitor.

Note that often merely opening up the CoCo will allow it to cool sufficiently better that no further manipulations are needed. A small table-top fan lightly blowing across the CoCo and the disk drives will also help greatly.

Many hardware hackers have removed their CoCos from their naitve cases and transplanted them into PC type cases. When this is done, a PC type, fan cooled power supply is used. While transferring the CoCo to a PC case is beyond the scope of this book, details are given in some back issues of “the world of 68’ micros” and FARNA Systems’ “Tandy's Little Wonder”.

CoNect sells a custom made case for the CoCo that uses a separate power supply. They will mount your CoCo in the case for you or take your system in trade for an already mounted system.

CoCo Keyboard Extenders:
Most computer users would prefer to have the keyboard in their lap and put most of the mess of their system off to one side of, or above, or below, their main work area. This is accomplished by adding an extension to the keyboard. An extension cable is available from HawkSoft that allows using the original CoCo 3 keyboard away from the computer.

The most elaborate and intriguing keyboard extender product is the Puppo Keyboard Adapter, sold by FARNA Systems. This device plugs into the CoCo mother board where the keyboard belongs, and allows you to plug in and use a standard IBM PC/XT type keyboard.

Now, “lBM type” keyboards come in a vast variety of styles, with great variation in key texture, tension, sculpting, travel, and positioning. This allows the end user to choose exactly what combination of these characteristics please him or her the most. Naturally, this IBM keyboard adaptor is the most expensive approach. But it may also be the best.

Monitors:
lf you are using your computer solely for text, the sharpest display possible will be that you will get using composite video output from the CoCo and a composite monochrome monitor.

Note that the IBM style TTL monochrome monitor will NOT work with a Color Computer! But some will want to avail themselves of the different color possibilities afforded by the RGB monitor.

RGB monitors for the CoCo will cost roughly 4 to 6 times more than a monochrome monitor. So you'd better decide you REALLY want color. If you do, the most cost effective monitor to get is the Magnavox 1CM135 (approximate cost: $270 new).

These monitorsoffer RGB analog, composite video, and RGB I (IBM CGA type) inputs, making them versatile. This, and their better resolution and brighter image and antl-glare properties make them a much preferred choice over the Tandy CM8.

Commodore monitors for the Amiga are similar to the 1CM135, and may be used with the CoCo. Monitors for the Atari ST can also be adapted for use with the CoCo, but you must invert the CoCo's sync pulses. The Atari puts out NEGATIVE separate sync, while the CoCo puts out POSITIVE separate sync.

When using the CoCo with some Sony monitors, you need to combine and invert its sync pulses. I do this in the Sony cables I designed for the CoCo using a single NOR gate on 74LS02 chip. All analog multisync monitors can be made to work with the CoCo if the right cable is constructed. For example, the older NEC Multisync monitors could be hooked up directly to the CoCo, by just matching the names of the input pins on the monitor to the output pins on the CoCo.

For those monitors that do not support audio, you may want to add a little speaker and amplifier and drive that from the CoCo's audio output.

More detailed hardware information can be found in FARNA Systems' book "Tandy's Little Wonder"

Telecomputing and OS-9

Bill Brady

Right away you may wonder about that title above. I mean, waht is different enough about OS-9 that telecomputing deserves a mention here? Well, there is a significant difference, and it goes to the heart and soul of the Operating System.

The Story so far...

First. let me dive into a very brief discussion of the current state of affairs. Telecomputing has evolved into two broad categories. The first is the host/terminal connection, the second is the bridge.

For the host/terminal connection, one device is assumed to have all of the computing power. This is the host. The second device, the terminal, is assumed to be an input/output device only. Inherently, OS-9 always assumes that it is the host and that the primary I/O devices connected to its hardware are all terminals. This can be seen from the line editing capabilities built into SCFMan, backspace-delete for example.

Bridges and Interconnectivity

The second broad category, the bridge, assumes -- even requires-- that computing power exists on both sides. Bridges are slowly emerging that allow the computer on one end ot the bridge to extend certain operating system and application features right into the computer on the 'other end'. Today, when a Macintosh computer and IBM PC compatible are bridged, the Mac an 'open a folder' and 'see' files inside -- icons and all -- even though that folder actually exists on the PC's hard disk! This is good in business applications where different types of computers often need to be connected together.

Bridges, such as Microwar's "UniBridge", exist today primarily to facilitate special functions; software development for process control computers, for example. In this instance, a developer creates software on a full blown computer system that is destined to run on a dedicated 'target' system.

Hosts and Terminals

We as personal computer users will soon become more interested in the bridge. For now, let's look closely at the host/ terminal interconnect.

Remember that OS-9 wants to be the host in this arrangement.

When we ask OS-9 to become the terminal, we are actually reversing the roles. This is done by running a program under OS-9 which makes the computer emulate in software the behavior of a hardware (dumb) terminal. Such terminal program sof tware re-arranges data flow within the computer so that this role reversal takes place. Data (usually just text characters) that would normally be routed from the host to a program are sent instead your screen. This part is wonderfully easy to do with OS-9 with its implementation of data paths and processes. This is why there are so many terminal programs for OS-9.

But there is one characteristic of terminals that is not as easy to 'emulate' under OS-9. It's difficult to emulate terminal hardware, dedicated to the sole task of being a terminal on an inherently multi-tasking computer. OS-9 is multi-tasking and multi-user; when you ask it to become a terminal it refuses to do only that no matter what you do. OS-9 is always looking for business to take care of elsewhere, for other users to service.

Making good terminal software

The second nut to crack is to create a terminal program that l) lets both OS-9 and the host think that each is the boss of the 'terminals', and 2) preserves OS-9 multi-tasking ability. Once you do 'crack that second nut' then a wonderful thing happens. You can have your cake and eat it too! To see the advantages of a "multi-tasking terminal program", let's look at the reasons that you may want to telecompute.

First and probably foremost, telecomputing opens to you a world of free (or almost free) information including software, digital pictures and music. These are accessible as files in other computers that you can 'read' or down load to your computer. Such files are found on telecomputing services such as Delphi, on local Bulletin Board Systems, or in a friend's computer system across town. To gain access all you need is a terminal program, a serial port, and a modem.

Here's the problem. Many of the files you will want are large and take a lot of time to download. When you run a terminal program on most computers (PC clones and the Macintosh for example) the download process completely ties up the computer. To be fair, the occasional terminal program allows you to do some work on your computer while you download "in the background". Even in these cases. the software places severe retrictions on what you can do while a file is downloading. For example, it may allow you to run that program's integrated text

editor but not your favorite word processor.

In contrast, with a properly written OS-9 terminal program, and windows, all you need do is pop over to another window and you can run ANY properly written OS-9 program of any type. [The extendable interface for WizPro allows the addition of any number of your favorite OS-9 applications to a WizPro menu. So, while you're online, just choose the application you need whenyou need it. This is especially ideal when using hard disks. They easily store all your programs and can load them into your CoCo on demand without haltingyour CPU. — PKW]

Conferencing by modem.

Another reason for telecomputing is conferencing. It is not only fun, but can be immensely informative. n a conference, messages from many people are combined and echoed back to the group as a whole.-lt's like a conversation where everybody can talk at the same time, yet the individual messages don't get mixed up, except ... you may have someone else's message pop up right in the middle of the one you are currently typing. This is easy to fix under OS-9. Your terminal program can create a separate 'device', or window, for you to type in. On WizPro, for example, this is called the Conference mode.

With conferencing, as with downloading, the multi-tasking ability of OS-9 again shows its differences. Say someone in the conference asks a question which, for you to answer it, requires that you list a file stored elsewhere on your system. With Level 2 and windows, you only need switch to another window to list the file.

How to start telecomputing.

At this point, you may be wondering how to get involved in telecomputing with your OS-9 computer. Here are some tips and information:

Serial ports: you want serial port hardware that will work with the standard port driver supplied with OS-9, AciaPak and STCio, for example. Don't buy port hardware that requires a special driver even if that driver is supplied.

Modems: Get a Hayes'm or compatible. Unless you know what you are doing and have a specific need to go faster, stick with a 2400 baud modem. And buy a modem/telephone line 'spike protector' at the same time as you buy the modem.
Terminal programs: here, a reversal applies. You want a terminal

program that uses a terminal oriented port driver, and not the one supplied with OS-9. Why? The standard drivers are designed to talk to a terminal on the other end... not to a host computer.

Commercial terminal programs: The best commercial terminal program is InfoXpress. This is available from Wittman Computer Products. This program allows one to set up a time and date to automatically log on and grab all your favorite forum messages, savethem to disk, and allow you to read and reply to them at a later time offline! Saves a lot of time and expense.

Shareware or freeware terminal programs: There are a number of good terminal programs that are either free or require only a modest fee. WizPro, a shareware program I wrote, is on the "Mastering OS-9" disk.

ln general, choose one that uses SAcia or a similar driver. Look for one, too, that has 'been around' for a while. Pay close attention to signs that it is being supported and updated. You will need a program that supports Xmodem since this transfer method is so widely used. Ymodem would also be nice.

Look for a program that has a built-in conference mode; if one you like does not include this feature, make sure it can be configured to read from and write to separate device windows which you set up yourself.

One problem you'll have with these 'free' programs is getting one in the first place. If you plan on downloading one, you'll need a terminal program to download the terminal program. A good plan is to log on to a BBS or service, and leave the following quetion: "Can anyone supply me with an address where I might send $10 or $20 and get a complete OS-9 shareware terminal program?"

I believe TelStar and XCom9, two popular terminal programs, are offered this way. Both work with SAcia.

Hard Drive Systems f or the Tandy Color Computer

Kevin Darling (Updated by F.G. Swygert)

OS-9 is a disk-intensive operating system. While this kind of operating system has many advantages, constant disk access can slow down a system. particularly on a Color Computer. Even with patches to speed up track-to-track access time, f loppies are slow compared to your CPU's ability to process data. Plus, your CPU typically stops doing any work while a f loppy disk is being read.

Two chief solutions to the disk access bottleneck are: buy a no-halt f loppv disk controller (this doesn't speed disk access but at least it doesn't halt the CPU); and buy a hard disk. Hard disks are shockingly quick and marvelously convenient.

Since many OS-9 users eventually investigate purchasing a hard drive system I asked Kevin Darling f or permission to reprint in "Mastering OS-9" some of his comments and advice on CoCo hard drives.

He addresses hardware and software issues in terms which may be bscure f or computer novices so I have made slight editions to improve its readability by beginners. — PKW

The basic parts of hard drive kits.

Hard drive system hardware is fast and intelligent. While there are some differences between hard drive systems, performance is excellent compared to floppies.

All hard disk systems consist of:

Power supply
Hard drive (s)
Hard drive controller
HCA - Host ComputerAdapter
(Hard Disk Interface Pak)

On other computers such as IBM compatibles, connecting a hard drive is relatively simple and elegant thanks to discreet slots in the back of the CPU case.

On the CoCo, most require a MultiPak Interface. All hard disk types discussed here involve an MPI except the Ken-Ton, which will work equally well with an MPI or Y cable.

Now let's discuss a few arcane details about hard drives such as how large the platters are, how fast they rotate, how much storage they provide, and so on. We also explain some of the abbreviations you may see in hard drive systems advertising.

Size — Most drive platters are either 5.25" (full or 1/2 height), or 3.5". As you might guess, full and 1/2 height hard drives take up the same space that full or 1/2 height floppies occupy.

Rotation speed and saf ety— Hard drives rotate at 3600 revolutions per minute (RPM). Compare this rate with your floppy drive's 300 RPM. The heads "fly" at 18 millionths (or less) of an inch above the coated aluminum disks (platters), which are sealed in a containment cover.

The extreme speed of platter rotation and the critically small distance between the heads and the platter make for a delicate machine. Do NOT move a hard disk while it's rotating. You can cause the heads to gouge into the surf ace of the platter!

Many drives have a safe landing zone on the innermost cylinders, thus some people use a Park command to move the heads-assembly there before shutting off their drive. Check around different information services for a Park command for your system if one is not provided. Most drives made over the past three to four years have an auto-parking feature, making parking programs unnecessary.

Advertising Lingo — What it means to you.
A hard drive is most often advertised as being one of three types: IDE, SCSI, MFM, or RLL. IDE drives won't work with the CoCo, so we won't bother to discuss them..

The MFM and RLL reference simply refers to the way data is stored on the drive. Both are older technologies that have been around for a long time. These drives will require a separate controller card. If you get an RLL certified drive, you will need an RLL controller to take full advantage of the added capacity. With a standard MFM controller, you will only get about 2/3 the RLL formatted capacity. Note that an MFM drive will format with an RLL controller and increase in size by 1/3, but it won't

od. May work fine for two or three months, then you boot one morning and get lots of errors!

The Small Computer System Interface (SCSI) was designed to support many different peripherals, not just hard drives. Macintosh computers use a SCSI port instead of expansion slots. SCSI drives have a controller built into them, meaning that a separate controller is unnecessary.

Data Storage Capacity

The bottom line for most users is the number of megabytes of storage a hard disk provides. The number of megabytes available for storage is not the only yardstick. Other measures of capacity are number of heads (or sometimes platters) and number of cylinders.

Usually, when formatted, you will use 32 sectors on each cylinder for each head. Each sector contains 256 bytes (the equivalent of about four lines of text on your monitor screen). How does all this add up?

Example: Suppose you buy a 10 megabyte hard drive. Suppose further that it offers 2 metal platters or disks. (It might also be advertised as a "4 head" drive because there is one head per platter side.) Chances are that the ad also mentions that the drive has 306 cylinders. As we noted above, when the disk if formatted you will have 32 sectors for each cylinder.

With 4 tracks each with 32 sectors, you have 4 *32 = 128 sectors available without moving the heads to another cylinder. Of course you CAN move to another cylinder — to 306 of them in this case. So for this example, (4 heads/cylinder * 32 sectors/head * 306 cylinders) = 39,168 sectors * 256 bytes/sector = 10 Megabytes.

Controllers -- General Operation

Hard disk controllers are very intelligent devices. They essentially have both a microcomputer and some RAM on board. They also contain digital to analog circuitry needed to control the drive. The microcomputer receives a simple command from the host computer (such as "read sector", "seek", etc.) and then interprets the command into the data it needs to carry out the instruction. This all happens incredibly fast. The RAM is used as a buffer to store one or more sectors of data. Thus the host computer is free to do something else while the controller does the hard work. This is very handy under OS-9, and many hard drive users don't bother with no-halt floppy controllers as the hard drive is in effect a no-halt device. Most controllers will handle two hard drives.

The HCA side of a controller resembles a set of addressable read/write registers. So an interface generally consists of an address decoder to place the interface within a certain range of CPU addresses ($FF70-77 is a favorite range for this). An interface also features a bidirectional byte-wide data buffer and buffers for Select, Read; Write, and Register Select address lines going to the controller.

In most regards the controller board is just like any other device that you might interface to your CoCo such as clock chips, PlA's, AClA's, A/D converters and so on. The HCA or interface pak would be different for an Apple, for instance. But the controller and drives could stay the same.

Note that I am NOT necessarily saying that you could drag your hard drlve setup from computer to new computer. You'd probably have to build an interface and write your own software.

CoCo Hard Drive Interfaces

This is the computer-specific part of the hardware. Each computer, depending upon its bus or ports, will use a different hardware method of interfacing to the hard disk controller.

The CoCo Hard Disk Interface available from Radio Shack is one example. It interfaces to a WD1010 controller (I suspect it might be useable with the WD1002 since Western Digital tries to keep its interfaces and software upgradeable). This particular interface is not a good one to try to use. The drivers are located in the OS-9 Level 1 Development Pak. They will only support a 10, 15, or 30 meg hard drive. They may work with others, but they were written for old drives available from Tandy.

The most popular interface for the CoCo was the Burke&Burke CoCo/XT and CoCo/XT-RTC. These are basically an interface adapter which converts the CoCo 40 pin bus to accept a standard eight-bit PC/XT hard drive controller card. Since eight-bit PC controllers and hard drives were some of the cheapest, this was a good way to go when it was first introduced. And it is still a very reliable system -- an excellent choice.

Several manufacturers built SCSI interfaces for the CoCo, including Disto, J&M, and LR-Tech. All of these are good systems, especially when used wit h a SCSI drive with built-in controller. Some were sold with SCSI controller boards that controlled MFM or RLL drives. These attach between the

interface and drive and are harder to package neatly.
The Disto is a small SASI/SCSI interface board that mounts in a Super Controller or RAMDisk. An adapter was available to mount the board in a MultiPak Interface. The only disadvantage to this device is that it would control only one drive at a time. An embedded controller SCSI drive can be used or a separate SASI/SCSI controller board.

The only interface currently in production is the Ken-Ton SCSI interface. This is a premium unit that will control up to six SCSI drives. It will also handle other SCSI devices, but drivers would have to be written for them. These are only available from FARNA Systems.

Power Supplies
Controller boards require 5 volts at around 3 amps maximum. Drives require 5 and 12 volts at varying amperage. Check specifications if you are "rolling your own". One thing to note is that drives can take up to 20 seconds to get to speed. During the first part of this powerup, the 12 volt amperage needed can be quite high. Don't skimp. A power supply with a fan is the best way to go to guarantee things stay cool and reliable.

My experience with hard drives.
I originally had a hard drive (19XS) that supported both OS-9 and DECB. Not many DECB programs worked with that drive, so I soon wished I had set it up entirely for OS-9.

Storage needs seem to grow. I started with a 5 meg drive, which seemed like a lot to me. lt was filled up sooner than I thought, so I went to a 10 meg. Nice try. Now l'm running a setup with both a 20 and a 10 meg drives, and still want more storage!

Actually, 20-30 megs should be adequate for most users. It is difficult to find small drives now, so get the best value for your money. Even though you may not need it, OS-9 will support up to 134 megs unmodified. It can be modified to support up to 4 gigabytes, but I don't know of anyone who has more than 350 megs on a CoCo. No need for that much storage unless you run a BBS!

Conclusion
This is just SOME information on hard drives. I hope it serves to answer your questions. Based on the information presented, you should be able to narrow down which drive systems you might be interested in picking up. Good luck!

Start BasicO9!

Dale L. Puckett

When Paul Ward asked us to write this appendix for his fine OS-9 book, we jumped at the chance to introduce you to one of the best kept secrets in the microcomputer world today — BasicO9!

The inspiration for this short piece was created long, long ago (nine years seems like an eternity in the computing arena!) in a galaxy far, far away. Its birth in Iowa verifies the latter part of our infamous introduction.

The revolution hasn't begun — it's already here!

lf you survey the magazines that serve the competitive personal computing world today, you'll find more than one rave review of True BASIC or ZBASIC as packaged for the IBM and Macintosh computers. Writers are heralding these new BASCs and proclaiming the birth of a new generation of programming tools.

Yet, the new features being strutted before an unsuspecting audience today are nothing new to seasoned OS-9 enthusiasts. They found these features in a revolutionary language from Microware Systems Corporation in Des Moines, lowa in 1979 and have been taking advantage of them ever since.

After you read this appendix, we hope you will be inspired enough to start BasicO9. Then you can take advantage of this language's state-of-the-art features too. After all, the price is right. Tandy includes BasicO9 free in every OS-9 Level 2 package it sells.

Once you start, where to go for help.

We often hear people with Color Computers say they don't use BasicO9 because it's too difficult to learn. Yet, they've never tried. To these people we say, "BasicO9 is not difficult. It's different!" After you run your first BasicO9 program and look back at your first modern BASIC code we think you'll agree that BasicO9 is indeed much easier to understand and use than the Microsoft BASIC interpreter built into the Color Computer 3.

If you are looking for a plain language introduction to BasicO9, we hope you'll pick up a copy of The Off icial BasicO9 Tour Guide. Commissioned by Microware, the book is a perf ect balance between completeness and accessibility. Highly

recommended. In it, you'll find a friendly, plain language introduction to this fantastic language and many examples. We also invite you to check out the many Color Computer 3 / BasicO9 programming examples published in the KlSSable OS-9 column in The Rainbow back issues and some of the articles currently in "the world of 68' micros" by Chris Dekker..

BasicO9 Advantages

When you start BasicO9, you'll discover for yourself that BasicO9 has many advantages when you write your first program. But since we don't want to keep you in suspense, we'll give you a sneak preview in this appendix.

First and foremost, BasicO9 is not the same old line-number-encumbered BASlCyou learned in school. Rather, BasicO9 is a modern programming language that closely resembles Pascal. ln fact, you'll find that translating most Pascal programs to BasicO9 is an easy chore.

Yet, while BasicO9 delivers Pascal's outstanding structural qualities, it is far less rigid. Since BasicO9 lets you create well-structured programs without line numbers, your programs will be easier to understand. You won't get lost following 15 "GoTo" statements-to meaningless line-number locations during your debugging sessions.

Speaking of understanding, you'll find that BasicO9 programs are very readable. While this may not seem important t-you now, it will be six months from now when you need to go back and change your program to incorporate new data.

And while you're writing or running BasicO9 programs you'll still have all the power of OS-9 Level 2 atyourflngertips. For example, if you forget the name of a file stored earlier, you need only type a dollar sign, $, followed by the word d ir and then strike the <ENTER> key to cause a listing of all the files in your current data directory to appear on the screen. You can do this from BasicO9's command mode and from its debug mode. Dir isn't the only command you can summon in this manner. You can run every OS-9 command in your current execution directory just as easily.

Level 2 offers yet another way to access your OS-9 command set. Simply strike the <CLEAR> key on your CoCo 3 keyboard. This will take you to another window where you'll find an OS-9 prompt waiting for you. The prompt is there because you started

an immortal Shell in that window earlier.

Easy to use AND Powerful?

I can almost hear the objection: “A programming language this easy to use can’t be very powerful!” Not so!

Look closely at a few of the commercial OS-9 Level 2 software packages you may have purchased. You'll most likely notice that several of them are stored in Basic09 I-code modules. This means they were generated with Basic09. Many of Bob van der Poel’s programs were written under Basic09.

You have access to the same programming language that commercial programmers use. And you received it virtually free with OS-9 Level 2.

Proof is in the Pudding

We’ve run out of room for the commercial already and haven’t even mentioned Basic09’s fantastic data typing capability. And since we wouldn’t want to tease you too much we’ll throw in a short example later.

But first, let’s see if we can come up with a few lines of code that prove Basic09’s claim to readability. We’ll start with a standard line-number BASIC program:

```
10 REM THIS IS THE OLD WAY
100 INPUT "PLEASE TYPE A NUMBER: ", X
110 IF X > 0 THEN 150
120 IF X < 0 THEN 170
130 PRINT "THE NUMBER IS ZERO"
140 GOTO 180
150 PRINT "THE NUMBER IS POSITIVE"
160 GOTO 180
170 PRINT "THE NUMBER IS NEGATIVE"
180 END
```

Now let’s do the same thing using a Basic09 procedure:

```
PROCEDURE OurWay
(* Sbow how BasicO9 control structures can make
(* your programs easy to read and understand. This
(* program  will produce the same results on your Color
(* Color Coputer screen as the program above.
DIM number: INTEGER
INPUT “Type a number: ”,number
PRINT
```

```
IF number > O THEN
    PRINT "Your nunber is positive."
ELSE
   IF number < O THEN
       PRINT "Your number is negative."
   ELSE
       PRINT "Your number is zero."
   ENDIF
ENDIF
PRINT
END
```

Enough said? I'll bet you love BasicO9 already! If you can still honestly say that the first listing is easier to understand after reading both, perhaps BasicO9 isn't for you.

If we may be allowed another commercial break, we typed the bottom listing with all lowercase letters. Later when we listed it, BasicO9
automatically typed its keywords in all UPPERCASE letters. It also automatically indented its control structures. All of this pretty printing is free. It's a bonus you get every time you write a program with BasicO9.

Data types and BasicO9 —
Something to write home about.

Earlier we mentioned BasicO9's powerful data typing ability. Now, we show you a simple example that demonstrates why this feature is something to rave about.

When you turn it on, BasicO9 knows five data types BYTE, INTEGER, REAL, STRING and BOOLEAN. Don't say it! You're right, that's no big deal since almost every BASIC today knows about at least three of these data types.

No, the big deal with BasicO9 revolves around another BasicO9 keyword, TYPE. First, to review:

* **A BYTE** is a data type that can be stored in a single memory cell in your computer; a cell is exactly eight bits or one byte.
* **An lNTEGER** variable is stored in a memory cell 16 bits or two bytes wide.
* **A REAL** number is a "floating point number", stored in a series of memory cells designed to hold floating point numbers (five bytes in BasicO9).

* **A STRING** variable stores English language characters and words that you can read on the screen. In BasicO9, strings can be any length if you have enough memory. You assign the amount of memory each STRING variable will use with a DIM statement.

* **And finally, a BOOLEAN** variable is stored in a single byte that can have only one of two values. A variable of type BOOLEAN must be either TRUE or FALE.

If you were forced to work with data in one of these five shapes, your universe would have a narrow scope and you would find it difficult to get anything done. No wonder many people hate BASIC!

You're more fortunate, however. You have BasicO9 with its unique TYPE statement on your side. If the readability examples above didn't win you over, perhaps an example or two featuring a few homemade Basic09 data types will do the trick.

Data types and a mailing list.
We'll begin with a real world example that could easily become the heart of a tool needed by most of us at one time or other. Imagine that you are the secretary of the local Lions club and you must set up a mailing list that you can us to print mailing labels and maintain information about everyone in the local club.

First, you must tell BasicO9 what kind of information you want to print on your mailing labels. Of course, we feel the best and one of the easiest ways to do this is to use a BasicO9 TYPE statement. Something like this should work handily.

```
PROCEDURE LionsLabels
(* Show how to use a BasicO9 TYPE statement
TYPE label=firstname:STRING14]; middleinitial: STRING[l];
lastname: STRING[20]; street, city: STRING[24]; state:
STRING[2];  zip: REAL
```

In the TYPE definition above we told BasicO9 to reserve 14 bytes to hold a members first name. Likewise, we reserved one character for a middle initial and 20 characters for a last name. Then, we set aside 24 bytes for the first two lines of our member's address, two characters for the member's state and five bytes for a REAL number to store his zip code.

In this particular example, we reserved 90 bytes of memory for each member of the club. Just remember, you don't have to use our definition for your mailing label. With BasicO9's TYPE statement you can have your labels your way.

Once we define our new data type, we must set aside memory to use it. We do this with the DIM statement. For example, if we have between 90 and 100 members in our Lions Club, we will probably want to reserve enough memory to hold the names and addresses of at least 100 members. This line will do the job:
DIM LionList (lOO): label

Here we have set aside 9,000 bytes of memorv to hold the information needed to print up to 100 mailing labels — 90 bytes per label times 100 labels.

To store our names and addresses in the array of l()() mailing labels, we can use any one of a number of techniques. ln all cases, we start with the knowledge that the name of our array IS LionList.

We also know that each element or member in our array of mailing labels has a number of fields. We defined these fields in our TYPE statement above. This means that the first mailing label in our list can be initialized with the following information:

```
LionList(l).firstname:=Dale LionList(l).middleinitial:=L
LionList(l).lastname:=Puckett LionList(l).street:=805 West
Edmonston Drive
LionList(l).city:=Rockville LionList(l).state:=MD
LionList(l).zip:=20852
```

Your array can be initialized with straight-forward assignment statements like those above. Or, you can use a standard loop structure to write information about all of your club members to the list at the same time. The loop can get the information from your keyboard or another disk file.

GET that data and PUT it here!

Once you have entered your data you'll be in mailing label heaven. You'll also be able to take advantage of Basic09's GET and PUT statements(Extended Color Basic, eat your heart out!).

For example, to print the first mailing label in your array to the

screen, all you need to do is use the following line:
PUT 1, LionList(l)
lf you've entered all the names and you want to print a hardcopy of your entire mailing list, all you need is the code below:

```
DIM printer:BYTE
OPEN #printer,"/p"
PUT #printer, LionList
CLOSE #printer
```

While we were developing our KlSSDraw tutorial series for The Rainbow, we used a similar technique with Basic09 data TYPEs and PUT statements in a drawing program. We called it KlSSDrawPut.

Because no data conversion is needed with the with the GET and PUT statements, we were ahle to substantially increase the speed of our program. With GET and PUT, an exact copy of the bytes in your structure is written to the screen.

GET and PUT also gave us a way to store pictures so we could save them to a disk file and reload them later. Increased speed came from the BasicO9 GET and PUT statements.

What is a line?

To show you how we can can apply the techniques used in the mailing list above to a drawing program, let's define and draw a line.

To draw an object that looks like a line we need a "tool" — a pen perhaps. Imagine a pen drawn on your computer screen whose pen point always marks where drawing can take place.

For now, we'll assume the line starts where the pen is resting. It runs to another location on the screen which can be defined by a horizontal and vertical pixel address. [A pixel is an element of your screen display which the computer can address. The word pixel is shorthand f or "picture element. " — PKW]

The first thing we need is a data TYPE definition for our line. Since we knew we also wanted to draw bars, boxes and circles, etc., we kept our definition generic. We defined a data type named Object:

```
TYPE object=DCode,HorP,VerP:INTEGER
```

Then, we reserved a place in memory to store it. We used the DIM statement and called our new variable Pen. It seemed like a good metaphor since we often draw with a pen in the real world:

```
DIM pen:object
```

We now had a place to store a pen we could use to draw a line. To use it we had to define our line and initialize it in memory.

Within OS-9, all drawing primitives are defined by the escape code, $1B followed by an additional byte. To put a line on the screen, we had to send $1B followed by $44. That means — in OS-9 speak — $1B44, followed by a coordinate pair, IS a line.

```
pen.DCode:=$1B44
pen.HorP:=100
pen.VerP:=50
```

We now have a line stored in a memory variable named pen waiting to happen. To make it happen, we must PUT it on the screen:

```
PUT #1, pen
```

Since we wanted to redraw that same line later, we needed to save a starting location. We named a new data type Orgin to complete the mission. We named our variable Handle..

```
TYPE orgin=DPSCode,HanX,HanY:INTEGER
DI Handle:orgin
```

The data field named DPSCode holds the OS-9 code required to position the data pointer on your screen, $ lB40. The fields, HanX and HanY store the starting location of our line. After we had reserved a place in memory, we stored the starting point for our new line:

```
Handle.DPSCode:=$1B40
Handle.HanX:=0
Handle.HanY:=0
```

We then drew our line with two lines of code.

```
PUT #1, Handle
```

PUT #1, Pen
Since that was too complicated, we designed a new data type to hold the starting location, the pen and the end point of our line. Since the two lines above drew a line on the screen when we ran them, Drawing seemed like a natural name for our new data type.

TYPE Drawing=Loc:orgin; tool:object

We named the field containing the starting point Loc (short for location) since that what it contains. Likewise we called the field that holds our pen a tool.

After we defined the objects we would be drawing, we needed a place to store our artwork. We called our work a Picture. It consisted of an array of Drawing(s):

DIM Picture(100):Drawing

This statement reserves 1200 bytes of memory for a picture made up of up to100 individual objects. Now, here's the magic. To draw your picture, all you need to type is:

PUT 1, Picture

Time to GET out of here.
What you've read here only scratches the surface. BasicO9 is indeed a powerful language. It is easy to use and it's fast. We hope you'll join us.

Syntax and Usage for the Stephen Goldberg Utilities Provided With "Mastering OS-9"

CLS clears the screen of your monitor. It can be executed from the command line by entering CLS or it can be called from a Basic-O9 procedure with the command 'SHELL "cls"'.

COPY uses exactly the same syntax as your original Copy utility that comes with your operating system (see your OS-9 Commands manual): COPY < source > < destination > [-s]

The enhancements include the ability to overwrite a file that has the same name as your chosen destination filename. You can now copy an updated file to the older version without first deleting the old one. lf you attempt to copy your file to one that already exists, you will be prompted with the question:

Rewrite? (y/n):

lf your answer is 'N', the copy will terminate at that point. If answered 'Y', the contents of the existing file will be replaced by the contents of your source file.

This version of copy will automatically delete a defective destination file. This can occur because of a read or write error or due to insufficient disk space. You will not be left with a faulty file to be deleted manually.

The last improvement is for single drive system users. It is a drag copying files from disk to disk with only one disk drive. The enhanced copy will reduce the disk swaps required by at least two exchanges.

Lastly, remember that you can increase the buffer size by using the command line modifier '#'. This will speed copying files and reduce disk exchanges. lf the buffer is large enough to hold the entire file, disk to disk copies can be made with just a single exchange on one drive systems.

COUNT counts the number of characters, words and lines in a text file. May be used with input redirection or in a pipeline.

COUNT [-opts] [filename]

The -1 option displays a count of the file lines.
The -w option counts the words in a file.
The -c option displays a file's character count.

Omit options to display all three totals.
lf no filename is present on the command line, the standard input path is employed permitting input redirection.

Example: count textfile <ENTER>
reads 'textfile' and displays the number of characters, words and lines contained in the file.

Example: list chapter1 chapter2 chapter3 ! count -lw <ENTER>
displays the number of lines and words in three files. lf no filenameparameter is entered with Count, lines can be accepted through a pipeline. All counts given when specifying more than one file given are totaed and the total is displayed.

Example: count -w -c < letter <ENTER>
counts the number of words and characters in the file 'letter'.

D displays an unformatted list of all filenames in your current directory, one filename on a line, if no oFtions are listed. The standard output of D is generally used to feed a plpeline.

D [-][name.option]

You can selectively display only those filenames desired by entering your criteria following D on the command line. You may use 'Wild Card' options in order to select the desired file names. The question mark (?) is used to denote a single character. The asterisk (*) indicates any number of characters.

The wild cards, * and ?, can replace characters at any position in a filename. For example, 'joker' may be represented as *r or *ker or j * or j *r or *k *. If you want to represent single characters you can use ?oker or j?k?r or ?????. All of these choices as a file-option will cause the filename 'joker' to be displayed.

The '-' option is entered directly before your file selection criteria with no space following it if you want all file names EXCEPT those matching your criteria to he displayed.

Example: d <ENTER>
displays the filenames in the current working directory.

Example: d ! sort <ENTER>
sorts the filenames in the current directory to the screen.

Example: d *.c ! count -l <ENTER>
counts the encoded files in your working directory. Your encoded filenames all end with '.c'. By using the option ' *.c', all filenames in your directory consisting of '.c' preceded by any number of any other characters (*) will be found and piped to count. The number of lines counted will be the number of encoded files.

Example: d ???? ! sort >/p <ENTER>
finds and sorts all four letter filenames in the directory and redirects the sorted list to the printer.

Example: d-*.c ! list <ENTER>
finds all filenames that do NOT end in '.c' and lists the files.

GREP searches the indicated file or files for the search string and displays all lines containing that string. The string will will be found regardless of the case of the string's characters.

To use, type grep followed by the string to be found then the name(s) or pathlist (s) of the file(s) to be searched then any desired options. If the filename is omitted grep will accept lines via the standard input.

If the search string contains a sace, then it must be enclosed in delimiters. You may use a quotatlon mark (") or slash (/) as a delimiter. Both start and close delimiters must be the same and may not be included in the search string.

The -c option causes grep to search for the string exactly as entered on the command line, matchin the case of the string characters exactly.

The -n option provides the fllename and line number of each line
displayed for easier location in the file.

The -t option displays totals of lines read, the number of lines containing the search string and the number of occurrences of the string in the file(s) searched after the display of lines containing the seach string.

The -o option displays only the totals not the lines.

The -v option causes only those lines that do NOT contain a

match to be displayed.

Example: grep jones addresses <ENTER>
will search the file 'addresses' for jones, JONES, Jones etc. and display all lines containing that string on the screen.

Example: grep -t jones file1 file2 file3 <ENTER>
searches three files for jones etc. and displays all lines as above. The -t option is used to obtain a count of lines and occurrences of the string.

Example: grep -o jones file1 file2 file3 <ENTER>
as above but does not display the lines, only the totals.

Example: grep -nc Jones personnel payroll <ENTER>
displays filenames and numbered lines containing 'Jones'. Only those file entries matching 'Jones' exactly will be listed because of the choice of the -c option.

Example: dir e letters ! grep 84/09 <ENTER>
will find all files in the 'LETTERS' directory which were last updated during September 1984. If no filename(s) are entered on the command line, grep can be used as a pipeline filter.

Example: grep /new jersey/ addresses <ENTER>
finds the people living in 'new jersey' in the 'addresses' file. Because of the space between 'new' and 'jersey', delimiters are required to define the total string.

Example: d ! grep -v afile ! grep -v bfile ! list <ENTER>
filenames of the current data directory are sent through a pipeline to grep which passes all but 'afile' to grep again in order to ellminate 'bfile' and on to list which lists all files except those deleted by the -v option. The output of Grep is via the standard output path and can be redirected or used to feed a pipeline.

Example: grep jones addresses > jonesfile <ENTER>
redirects the output of grep to 'jonesfile' on disk.

Example: grep 11714 addresses ! sort > /p <ENTER>
prints a sorted list of those living in the 11714 zip code area.

SORT is an in-memory ASCII sort for files, directories etc. The input and output of sort are via the standard paths and can be redirected or sort may be used as a pipeline filter.

The sort buffer is approximately 3.5K. Its size can be increased for larger files with the use of the '#' command line modifier. If the sort buffer is not large enough to accomodate all of the linesto be sorted, the sort will abort and a buffer overflow notice is sent out the standard error path. You may then re-sort with a larger buffer. The size of the file to be sorted is limited only by the memory available.

Example: sort < namelist <ENTER>
displays on screen the sorted entries of the file 'namelist'.

Example: sort < address > sortedadr #30k <ENTER>
sorts a large address file to the diskfile 'sortedadr'.

Example: grep jones address ! sort > jonesfile <ENTER>
finds the addresses for all the 'jones' and sorts them alpha betically and saves the sorted list on disk in 'jonesfile'.

Sort can also be used to sort entries from the keyboard: sort <ENTER>. Type in the list to be sorted. When done, you press <ESCAPE> (<CLEAR> <BREAK> on the CoCo) and the list will be sorted.

You may redirect the sorted output of the keyboard entries: sort > /p <ENTER> or sort > /d 1 /newfile <ENTER>. This permits printing or saving to the disk your sorted keyboard entries.

UNIQ [-c] deletes repetitive entries from sorted lists. The standard input and output paths permit its use as a pipeline filter. The '-c' option provides a count of each sorted item. Only one of each duplicated line is displayed staring with the count.

Example: sort <namelist ! uniq <ENTER>
sorts all names in the file 'namelist' and Uniq passes only one copy of any duplicated names.

Here is a simple example of the use of the '-c' option. Each time a customer places an order with you, you add his name to a 'customer' file. At the end of the month, you would like to see which accounts are most active. The file contains:

Smith Jones Smith Williams
Roberts Peters Jones Smith
Williams Smith Williams Jones

Sort the file and pipe the sorted list to uniq, using the '-c' option and the sorted list looks like this:

```
sort < customers ! uniq -c <ENTER>
3 Jones
1 Peters
1 Roberts
4 Smith
3 Williams
```

The standard output of uniq can be redirected to another file or to the printer.

OS-9 and Music

Michael J. Knudsen

One of human kind's most glorious and mysterious activities is making music. Music can be at turns explosive and serene, calming and terrifylngly transforming.

The subtleties of music perception fascinates many music professionals and educators. colleges and universities spend increasing amounts of time measuring the way we hear and understand sound. Computers and music synthesizers are ideal tools to aid this pursuit.

Two educators interested in music perception are Jack Taylor, professor of Music at Florida State University since 1970 and Steven Newcomb, also at FSU. During the Seventies, Taylor's interest in musical applications of digital technology led him to organize conferences around the country. At these conferences researchers and educators kept abreast of developments in the field. One topic of chief interest at these conferences was electronic music synthesis which naturally attracted synthesizer pioneer Sherwin Gooch. Taylor and Newcomb developed a reationship with Gooch who donated to Taylor and FSU an advanced synth and composition device named Platypus.

In 1980 he and colleague Newcomb founded the pioneering Center for Music Research at FSU. One of the first of its kind, the Center developed technological approaches to studying music perception — and to studylng music in eneral, including composition.

"The timing was right at FSU," says Newcomb. "We had a new dean who was receptive to the idea of usin the latest digital technology." This support along with Gooch's gift kicked off the Center to a great start. Now, the center boasts dozens of custom digital synthesizers and an 30-terminal system for student use.

OS-9 is central to that terminal system. How it came to be central to CMR's work is ironic: one of the CoCo 3's competitors in the entry-level computer market, the Atari ST, introduced Newcomb to OS-9. Newcomb wouldn't even have gone looking for OS-9 for the ST except that the existing software and operating system

performed poorly.

According to Newcomb, he chose Atarls “because they have the biggest bang for the buck. The problem was that GEM just doesn’t work.” GEM, the Graphics Environment Manager, offers a point-and-click environment for the ST series that has sadly been plagued with problems since its introduction. “We also found the hard disk interface to be unreliable and it’s just too hard to back up floppies,” adds Newcomb. Plus, GEM and existing ST software didn’t offer the programming maturity the Center was looking for. The purchase of STs was beginning to seem like a mistake.

Then, while Newcomb attended the 1986 X3D1.8M ANSI conference he received a call from David Kaleita, then president of the OS-9 Users Group. “Dave said, ‘You know, OS-9 is coming out for the ST soon.’ I decided to try it,” Newcomb reports. He bought Personal OS-9 for the ST and ran it through some paces. He was astounded. OS-9 was software of “incredibly high quality”. Even more, it performed in a UNIX-style fashion. Since CMR had just installed a Sun workstation running UNIX for students and researchers, using OS-9 meant an easier learning curve for students.

Since starting with OS-9, Newcomb has developed a music software writing curriculum based on the OS-9 C compiler, which Newcomb admires. “It has the best error messages of any C compiler, period.” Because it is exacting in syntax it’s a great educational tool. “I like it because it’s persnickety,” laughs Newcomb.

When asked about why OS-9 was a good choice for CMR, Newcomb said, “OS-9 gives us a reliable platform on which to do serious work. The operating system always does what it’s supposed to do.” Still, OS-9 doesn’t carry all the weight at CMR. Some GEM-based applications are still used and their new Macintosh lI gets a workout.

One problem with developing OS-9 based musical applications on the ST, Newcomb notes, is that the MIDI port built into the ST is not currently implemented under ST OS-9. The problem is that “the keyboard, mouse and the MIDI port all use the same interrupt level.” This hurdle is about to be jumped, says Newcomb, without disclosing details.

OS-9 and music have gone hand in hand for longer than one might expect. Years ago Eric Miller of Microware developed MIDI sequencing software for OS-9 which he reports is "rock solid" and completely suitable for live stage performance. David Kaleita also uses OS-9 sequencing software when he composes on his own synthesizers.

Will advanced OS-9 sequencers ever be made commercially available? There are rumors, especially from Florida, that OS-9 may suddenly jump into everyone's musical consciousness very soon — but further details are confidential.

Before we move on to our discussion of OS-9 musical aplications on the CoCo, it should be noted that one of the most advanced series of musical workstations, the Fairlight, has run OS-9 for as long as the 6809 CPU has been around. The first Fairlights used OS-9 and several 6809s. Now Fairlights come equipped with 68000-family microprocessors, still running OS-9.

In olden times, a synthesizer converted voltages into notes. To get two synths to work together was difficult and crude. One synth could force another to play along with it by transferring voltages corresponding to its notes to the other synth.

A wide varlety of technical problems, including the inability to alter each synth's patches through one keyboard, discouraged synth connectivity. Often it was more trouble than it was worth.

But it worked. It was such a tease to be able to "layer" your synth sounds that the electronic music industry responded. Synth manufacturers, led at first by Roland and later fully supported by dozens of others, developed a standard computer code for inter-synth communication. This standard code allowed synths of ALL manufacturers to communicate note and "volume" (key velocity data as well as patch change, pitch change, and so on.

This code — the Musical Instrument Digital Interface Specification 1.0 (MIDI) — has held sway for over half a dozen years. It has caused a revolution in how synths and other electronic devices speak to each other. One keyboard can have fine-tuned control over an entire multi-keyboard setup. This allows a musician to provide musical nuance and technical flexibility undreamed of ten years ago.

MIDI consists of digital data. This is what computers eat for breakfast. And computers can produce MIDI data too. This allows musicians to use computers for composition and arranging. Musicians can alter the tones a synth produces by moving a mouse around their computer screen. Entire keyboard setus involving dozens of tones and digital compositions can be initiatec with a button press.

Sound exciting? Sound difficult to do on a Color Computer 3? Exciting, yes; hard to do on your CoCo, no. MIDI data is transferred at over 31,000 bits per second. This is 31 kilobaud, or simply 31K baud. While this may seem extremely quick (it is!), most computers — even the older Color Computers at 1 MHz — can easily handle the transmission rate. The Color Computer 3, at 2 MHz, makes the job even easier. In fact, one first step has been completed by a programmer and CoCo aficianado, Michael J. Knudsen. Knudsen has had a MIDI sequencer for the CoCo 3 called UltiMusE III out for some time now. This package is derived from his shareware program UltiMusE available for free download on Delphi. There is also an OS-9/68000 version available for the MM/1. Both packages are currently available from Northern Xposure.

Few people are as knowledgeable about MIDI programming on the OS-9 CoCo 3 Knudsen, so I turned to him to find out his perspectives on MIDI and the CoCo 3. Here is a summary of comments he left on Delphi:
MK: (on whether a MIDI product f or the 8-bit CoCo can compete with Ataris and Amigas): First of all, MIDI data streams in all 8-bit bytes. This makes 8-bit computers a natural. In fact, a 16 or 32 bit micro [as in the Atari and Amiga] is overkill. Being the best of the 8-bitters, the 6809 is plenty ood enough for MIDI work. The 6809 has predictable instruction executlon times (given the clock frequency) so is good for timing loops. It has three interrupt inputs so can handle real-time events well when recording keyboard performance on MIDI.

PW: Supposeyou're trying to record MIDI data intoyour CoCo under OS-9. The OS-9 operating system takes such control over the interrupts maintain multitaskin; how could it keep up with 31 Kbaud data input?

MK: OS-9 systems, ateast the CoCo, have a hard time recording MIDI and time-stamping events because the interrupts are not as available to the programmer as they could have been (thanks to the Multi-Pak, mostly). Having a real-time millisecond clock

chip would help. But given some external hardware aid, such as a real-time stamper and buffer such as a Roland MPU-401, MIDI recording under OS-9 is a snap.

I want to point out that OS-9 is already excellent for MIDI playing, as the sleep) call can time between note on/off events to 1/60th second accuracy.

PW: Still, what would inspire a programmer to write a MIDI programming for an 8-bitter like the CoCo 3?

MK: Multi-tasking. OS-9's multi-tasking is an untapped gold mine for MIDI. You could have a score notation player (like UltiMusE) in one window, a,patch librarian for your synthesizer in another, and a recording sequencer ln a third.

The neat thing is that none of these programs have to be designed or coordinated to work together (as they woud on a PC or ST). You just move among different windows to get the one you need at the moment. Tweak your synth patches in one window, then go play your score in another.

PW: How does this differ from what electronic musicians are doing now ?

MK: There's a big difference. Already MIDI musicians are wishing they could patch all their equipment from one common computer screen. On any other OS, you'd have to dump your Yamaha patch librarian and boot your Roland one and so on. But under OS-9 you can have them all in different-windows, maybe several on one screen. This will be especially nice on a screen with resolution such as on the Atari ST. Maybe when OSK-ST [Personal OS-9/68000] gets a CoCo 3 type windowing system we'll get this. But, of course, it's easy to do on the CoCo 3, too!

PW: Why have there been so f ew MIDI programs f or the CoCo ?

MK: No good reason. The CoCo has a lot of catching up to do in the MIDI software race — the lowly Commodore 64 has been way out ahead of us for years. But OS-9's multi-tasking could turn out to be our secret weapon to get musicians into OS-9 computers. I'm proud to be the first to contribute music software for OS-9/6809, but I hope I won't be the last.

COCO-3 BOOT LIST ORDER BUG (BLOB)

Facts, fixes and theories

Kevin Darling & friends

Some owners have it, some have never seen it. Ordering of modules in a bootlist for os9gen seems to affect it. Adding new devices may cause it to show up. What causes it? It's past time to lay out both what has been conjectured and what is truly known so far.

At first, the OS-9 kernel itself was blamed. We've been pretty sure now for a long time that it is NOT at fault. All the modules are position-independent, and have been gone over very closely by several of us, looking for anything that could cause a problem. We have found no software cause at all (with the exception of the disk driver - see below).

Instead, hardware and timing discrepancies in the CoCo-3 and peripherals have been found almost always to be at fault. In fact, it's often possible to pinpoint the exact cause of a particular problem, with enough information.

Enough preliminaries. Here are most of the confirmed and unconfirmed symptoms and possible reasons, including things that act like BLOBs...

FLOPPY FORMATTING HALTS IN FIRST FEW TRACKS; READ/WRITES ARE OFF BY A BYTE:
Ken Schunk, myself, and others long ago found that the halt method used by CC3Disk (and some DECB drivers in programs) has a problem with some disk controllers (apparently mostly pre-1985 1773's). The usual method is to wait for the FDC (floppy disk controller) to indicate it is ready to exchange a byte of data, and then have the CoCo go into the halt mode. What will happen is that the first byte transfer gets lost, and this is returned as a "Read Error" by the driver.

For reasons as yet unknown, this "data lost" sequence sometimes "seems" to be driver position dependent. I would guess that most boot failures are caused by this one, especially with older controllers (although I've seen it happen on newer ones, too). The drivers can be fixed, and we should be able to post patches later.

READS/WRITES GO TO WRONG LSN:
Actually, they go to the wrong TRACK, which is also always the wrong LSN. Usually caused by using disk drives that are set to turn on their motors only with drive select, instead of the required method of all motors on with the motor-on signal. All drivers assume that if one motor is on, ALL are on. Because of this assumption, and especially because the drive READY line isn't usually available on the CoCo setup, the FDC will send stepping commands to a drive that is still spinning up again when selected (it takes about 1/2 second to be actually "ready").... and those stepping pulses are totally ignored by drives not spun up. So while the FDC _thinks_ it's stepped the head to a new track, in fact either some or all of the step pulses have been lost. Worse, the 1773 FDC seems to ignore the imbedded track information on the disk itself (contrary to docs) and so as long as the sector number matches up, the data is read/written... to whatever track the head happens to be over!
So make sure your drive motors all come on at the same time.

SPEED AND BAD CHIPS:
Testing and experiences by several people has shown that the American semiconductor industry has gotten pretty bad over the last few years as far as quality goes. Or perhaps retailers are selling more reject chips that they buy on the grey market. In any case, some failures of chips used in add-on devices have been found to be brand dependent.

For example, some of the LS245 data buffers inside CoCo-3's seem to fail to pass true data at times. Replacing this chip with a Japanese brand will usually cure this particular problem. Motorola chips seem to be the worst bet. Symptom is that an instruction loop reading from the MPI sometimes sees bits set that it shouldn't. Solution is to replace the chip or slow down the loop.

Speedwise, many people use hardware designed and built for 1Mhz operation from the CoCo1/2 days. A common problem is with RS232 paks... they may need the 6551 replaced with a higher speed version.

INTERRUPTS:
Boot problems also sometimes appear when a device's interrupt line isn't correctly reset. I've had several 6551 ACIAs (used in RS232 paks, etc) that decided not to clear their interrupt line just by resetting the CoCo. This leaves an interrupt hanging and can mess up a machine trying to boot OS-9.

It's also been found that some RS232 paks were built with the E clock tied to the IRQ line... this can abort a boot also.

Stuck interrupts are covered in the various "IRQ HACK" files available on most networks, as are files on the RS232 pak.

MULTIPAK UPGRADE:
A non-upgraded MPI definitely causes problems. At the least, it can cause wrong information to be read from the crucial GIME interrupt status port.

The most common rumor we see on BBS's is that the MPI upgrade "isn't needed", because "my machine runs fine without it". DO NOT LISTEN TO THESE PEOPLE.
PLEASE EXPLAIN TO THEM THAT THEY ARE STUPID.
While we can't swear that you WILL hurt your GIME if you don't upgrade, we can certainly say that it does make electronic sense to DO the upgrade (plus Tandy sold the upgrades at first cheaper than their cost, which alone would make one think there's a good reason for having it, eh?).

The electronic reason for the upgrade is this: a READ from $FF80-9F will turn on BOTH the GIME data bus AND the MPI data bus. (In addition, really old MPIs ghost their slot select at $FF7F and $FF9F, which causes problems.) It's never a good idea to have two devices trying to put data on a bus at the same time... one of them could get hurt (usually the GIME, in reported experiences). Especially under OS-9, where the interrupt register at $FF92 is read at least 60 times a second, it makes sense to not have that data be corrupted by bogus MPI data coming on at the same time. So UPGRADE YOUR MULTIPAK !

E-CLOCK SYNCHRONIZATION:
All accesses to peripherals need to use the 6809 E clock to validate the transfer of data (especially at 2Mhz!). A few early versions of third-party devices accidentally were made with registers that didn't do this. All have been fixed for a year now, as far as I know.

The boot-order side of this came about whenever a device register was accessed at an odd/even address, and then the next cpu instruction fetch was at the opposite even/odd address... which meant the A0 address line (or sometimes A1 and maybe A2 also) would change after the E cycle ended and thus cause wrong device register addressing. This was shown on scopes as a small (around 10-ns) glitch.

So the *position* of the driver I/O access instructions in memory was very important, and was a true common “boot order” trouble causer (and may still be with older devices made in the pre-CoCo3 days).

GIME S0-3 DECODING:
A variation of E-gating is that the SCS external select line is generated inside the CoCo-3 without being E-gated. This could possibly mean that while the GIME is decoding a different I/O selection, the S0-2 GIME lines decoded by the 74LS138 in the CoCo could easily wobble between outputs, possibly randomly enabling ROMs, PIAs, etc and placing bogus data on the bus. It also may be one cause of the video “sparklies”.

Again, using the E gating on devices should mostly solve this, altho it’s also recommended that if you have problems you should gate the 138 with the E clock (Roger Krupski came up with the easiest method: inside the CoCo on the cartridge port, simply tie the E clock to the SLENB line.

DOUBLE INTERRUPTS:
This is an oddball one. Sometimes people notice that their boot fails, or that their software clock runs at double speed while within a VDG screen. Quite by accident, I stumbled across evidence that certain address bit combinations in these situations causes double the vertical interrupts to be generated. No solution except to boot to a real window always, and if you have this clock problem to change the order in which you start up a game, so that it’s video address can be moved somewhere “safe”. This also seems to be GIME dependent. Non-upgraded MPIs can cause this also, I think.

OTHER HARDWARE PROBLEMS:

Bad connections. Bad connections. Bad connections. Clean all your contacts regularly. The cartridge port, the MPI and slot pins, all rompak devices, disk drive cables, and even yank your GIME and swab it with alcohol if need be, although sometimes just pushing/tapping on it cures many oddball troubles. Make sure your drives don’t have something covering the write-protect detect LEDs. In general, just keep everything clean!

It’s also about now that many disk drives in use for years, are wearing out or becoming misaligned. Heads become a lot weaker, and data becomes flaky.

We've also seen cases where a new cordless phone, or appliance on the same circuit breaker, can screw up floppy or hard disk transfers. Even satellite dish downfeeds running by the computer. If you start to have problems, ask yourself "did anything change here lately?"

OTHER SOFTWARE PROBLEMS

More and more often, we find that many supposed boot list problems often have an unrelated simple explanation... such as making a new boot and forgetting that you patched some modules or used old ones; the common "oops forgot to put Grfdrv and Shell in the CMDS directory" gotcha; leaving out a module. Very often it can be caused by not having the latest drivers for a device. It's important to keep updated with the newest software made available.

Also, sometimes a module (especially os9p1) will get hit by an errant program, and then you os9gen a new disk... which gets perpetuated with the bad os9p1 from then on through new os9gens. We also find that people often reverify a bad module quite by accident using disk editors on their bootfile, thus hiding future trouble. Keep a log of all changes you make, and CRCs!

MISC THEORIES

Most other problems fall into the mystery section (meaning we don't have a firm handle on the cause yet). I have two pet ideas that may or may not make sense, but which are bolstered in part by experiences by myself and others.

One is that since interrupts cause the internal BASIC ROM to turn on (to get the interrupt vectors), the ROM stays on a bit too long and corrupts the data bus at times. Probably a dumb theory.

The other is that the dead cycles within many instructions have an effect. During the dead cycle the address bus contains $FFFF (which turns on the ROM!) and again, perhaps this data sticks around, or the address lines change too fast enough once in a while from true address to FFFF. This ties in with partial evidence that some 6809s at 2Mhz will start changing their address lines immediately after the end of an E cycle, perhaps even before E-gated devices finish up. We do know that oddball reads/writes occur at times to strange addresses, and this might explain them.

A third theory gaining some acceptance (but we just don't know

how the GIME works internally) is that the GIME, like the SAM chip, powers up using either the up or down side of the main oscillator clock (remember hitting reset on SAM machines to get the right red/blue fake color phase? like that). Perhaps one side is better than the other. Certainly powering down sometimes cures a boot or other problem. So who knows? We also know that changing cpu brands, and sometimes switching GIMEs, will often cure timing problems and the sparklies. Not always, though.

FAULTY DISK CONTROLLER CHIPS:
Replacing the 7406/7416 chips in older floppy disk controllers with a different brand can help. Three people have called lately with info that some of the Fairchild chips have nasty waveforms.

CONCLUSIONS
We're still gathering data, and occasionally do run across something unexplained. For the most part though, BLOBs have become fairly rare. This may be because people have more L-II experience, or newer hardware, or a combination.

OS-9 itself is not at fault, and note that even DECB applications can and do suffer from the same symptoms. The basic answer is that we moved up to a faster machine, while still using older peripheral equipment.

The order of the bootlist CAN affect the symptoms (as we've seen), but this is simply software showing up hardware bugs, and is NOT the fault of OS-9 itself. So the final word is this: our best evidence is that there really isn't a boot list order bug. Look to your hardware instead.

The above information has been gleaned over the past two years from personal experience, many phone calls and network messages, and the work of Bruce Isted, Tony DiStefano, Chris Burke, Roger Krupski, DP Johnson, Dave Wiens, Ken Schunk, and many others.

NOTE: A complete set ofpatches known to cure most BLOB problems are available from Sub-Etha Software.

MISCELLANEOUS BLOB TIPS

Michael Shell

Here's a bunch of odd ball items that you might find helpful:

1. In case anybody is wondering who turns up the speed, one of the last things the boot module does is to put the CoCo into the 2 Mhz mode.

2. If you have test code within a device driver and you want it to send you a "signal", try poking values into the border register at $FF9A. OS-9 won't mind a bit.

3. If your disassembler is the type that misses IRQ routines and printsthem as a bunch of fcbs, try this trick:

a. Disassemble the code.

b. Put in long branch subroutines (lbsr) to the beginning of each IRQ routine that the disassembler missed. The best place to do this is in the last instructions that the disassembler picked up:

```
decb
bne L004b
clrb
rts
* end of code here
```

Becomes:

```
decb
bne L004b       note that the lbsrs are in a "path of
clrb          execution"
lbsr D0100      branch to IRQ#1 that disa missed
lbsr D02f0      branch to NMI that disa missed
lbsr D01a0      branch to IRQ#2 that disa missed
rts
* end of code
```

c. Assemble the code with asm.

d. Disassemble the resulting object code.

e. Remove the statements that you added. Remember that the offset labels to code after your inserted code will no longer be valid as offsets from the start of the module. If you have no labels after the place where your lbsrs were, then this will not be a problem.

f. Presto, fully disassembled code!

3. The magnetic zones on a disk tend to repel or attract each other depending on their respective polarities, just like little magnets. Thus, the ones and zeros on a disk tend to "move" a bit depending on the adjacent bits. If pronounced enough, this effect can lead to read errors. This is particularly true on the inner (high #) tracks where the bits are closer together. Write precompensation is a process in which the position of the bits is shifted during writes to compensate for the natural "wandering" of the bits. As Kevin Darling mentioned in his book "Inside OS9 Level II" (p 3-5-3), cc3disk never activates the 1773's built in write precomp circuits. However, I found unused, leftover code in cc3disk that is designed to activate write precomp. Talk about skeletons in the closet! Perhaps future patches could renable precomp on the inner tracks. A small improvement in read relability could be realized especially by those using older drives.

4. If you are experiencing I/O errors with a MPI, especially with the buffered mode of the SCII, try changing IC1 on the older MPI or IC2 on the newer MPI. These chips handle the E clock and some other signals. I have seen a case in which a bad chip caused some of the address lines to bounce with transitions of the E clock -> bad news!

5. The owner's manual of my Disto SCII has a mistake in table 2. The register addresses are listed backwards. Here's the corrections:

Table 2 - SCII registers

Location Hex	Location Dec	Description
FF77	65399	FF76 mirror
FF76	65398	write: D0 = 0 FDC write operation *1 = 1 FDC read operation *1 D1 = 0 normal mode = 1 buffered mode D2 = 0 normal NMI = 1 masked NMI D3 = 0 no FIRQ (masked) = 1 FIRQ enabled read: D7 = FDC INTRQ status (inverted)
FF75	65397	FF74 mirror

FF74 65396 Read/Write buffer *2
*1: In the buffered mode.
*2: Any write to $FF76 or $FF77 clears the buffer counter.

Note that the data buffer mirror allows std and ldd for pseudo 16 bit transfers.

Also, there are a few corrections needed to the SCII schematics:
On V1.3:
a. There should be an inverter in series with the line that leads to the counter (between pin 18 of U6 and pin 1 of U7). This is the small “hack” that you see on the “wrong” side of the PC board near the CoCo connector.

b. The address line pins of U6 should read: 1,3,5,7,2,4,6,8 not 8,7,6,5,4,3,2,1 as given.

On V1.4:
c. Pin 1 of U8 should be shown connected to GND not VCC.

On both versions:
d. Pin 10 of U8 connects to pin 12 of U13, NOT to pin 13 of U13.

6. Concerning GIME S0-S2 hacks: Four signals are produced by IC9 on the CoCo3 motherboard. They are SCS (which helps decode I/O devices), CTS (which turns on the cartridge ROM), ROM (which turns on the motherboard ROM), and the signal which decodes the PIAs which I’ll call PIA. These signals belong to two different groups.

a. Address Decoding Signals (ADS). SCS and PIA belong to this group. These signals are used in conjunction with the address lines to decode an address. The E clock is then used to actually turn on the device.

b. Chip Select Signals (CSS). CTS and ROM belong to this group. When these signals are active, the devices turn on regardless of the E clock.

The problem many hackers see with all this is that CTS and ROM do not APPEAR to be gated with E (unless the GIMEs do this internally). So, a ROM device could start outputting junk onto the data bus without checking E. Since ROMs tend to be a bit slow, this could cause a collision on the data bus. The result is a BLOB causer. As a solution, many hackers rigged U9 so

that ALL FOUR signals were gated with the E clock. This is fine and well for CTS and ROM, but I don't think this is a good idea for SCS and PIA. The ADS signals should be stable BEFORE E becomes high in order to give the address decoding logic enough time to stablize prior to E becoming high. If SCS changes with E, a logic race condition could occur in some CoCo accessories. This could cause I/O glitches. So, I suggest that any E gating hacks should be confined to CTS and ROM. A 74LS157 (or 74ACT157 if you want the best) should do the job nicely (E clk to A/B select; A inputs go to VCC; B inputs go to CTS and ROM; outputs are new, gated CTS and ROM).

Since CTS is hardly ever used under OS9, but ROM is used for every interrupt, you can try a simpler hack that affects the ROM on the motherboard only. Using an inverter, make an inverted E clock and then route this signal to pin 22 (OE, active low) of the ROM. Be sure and remember to cut the trace that currently grounds pin 22.

As a final note, all of this address decoding appears to have been done correctly with the SAM chip in the CoCo2. However, without knowing how the GIME generates S0-S2, we can't be sure that it is done correctly in the CoCo3. Note that the GIME probably does SOME additional processing of these signals. For example, R/W is probably taken into account to prevent the CPU from trying to write to a ROM and thereby creating the mother of all bus collisions. If the GIME does indeed consider E and R/W properly in developing S0-S2, then no hack should be needed at all. Gating SCS with E, as most "improper" hacks do, delays the enable of the 1773 chip by a few nanoseconds. This could affect the 1773 BLOB problem. Thus, many S0-S2 hacks may appear to fix GIME problems, when in fact, the 1773 was the problem all along. Time spent with a logic analyzer could settle this issue once and for all.

NOTE: A complete set ofpatches known to cure most BLOB problems are available from Sub-Etha Software.

OS-9 Users Group M embership Information

(data accurate as of 04/15/95)

The OS-9 Users Group is an international non-profit organization devoted to exchanging and distributin information about, and public domain software for, all available versions of the OS-9 Operating system.

The OS-9 Users Group periodically publishes a newsletter entitled M OTD (Message of the Day) which contains many useful articles, software listings, and other information helpful in keeping OS-9 computing enjoyable and rewarding. Other membership benefits include free technical help referrals (by mail or electronic BBS) and significant discounts on the purchase of individual volumes of the OS-9 Users Group Public Domain Software Library.

One year memberships in the group cost $25.00 and includes a one year subscription to the MOTD newsletter and the right to purchase additional disks of software at a very reasonable cost.

The group's public domain software library currently has over 56
individual volumes of software comprised of almost 300 individual programs. The library is constantly growing due to the group's policy of sending one volume (disk) from the library free for each individual program donated by a member.

For more complete information on the OS-9 Users Group, including a complete catalog listing of (and ordering information for) all currently availIable volumes in the Group's public domain software library, visit the OS-9 Forums on either the CompuServe or Delphi electronic networks.

To join the OS-9 Users Group, send a check or money order to:

The OS-9 Users Group
6158 West 63rd Street, Suite 109
Chicago, IL 60638
USA

Memberships run from 01 January until 32 December. Send an SASE for a membership kit which will include a membership form and more information about the User's Group as well as a pro-rated price for joining the Group in mid-year.