

 Osborne/McGraw-Hill

THE OSBORNE/McGRAW-HILL GUIDE TO YOUR

Apple[®] III



Stanley M. Miastkowski

THE OSBORNE/McGRAW-HILL
GUIDE TO YOUR

APPLE[®] III

THE OSBORNE/McGRAW-HILL GUIDE TO YOUR

APPLE[®] III

Stan Miastkowski

Osborne/McGraw-Hill
Berkeley, California

Published by
Osborne/McGraw-Hill
2600 Tenth Street
Berkeley, California 94710
U.S.A.

For information on translations and book distributors outside of the
J.S.A., please write to Osborne/McGraw-Hill at the above address.

Apple, Silentype, and the Apple logo are registered trademarks of
Apple Computer, Inc.

Apple Unifile and Apple Profile are trademarks of Apple Computer,
Inc.

The Source is a service mark of The Source Telecomputing Corporation.

THE OSBORNE/McGRAW-HILL GUIDE TO YOUR APPLE® III

Copyright © 1983 by McGraw-Hill. All rights reserved. Printed in the United
States of America. Except as permitted under the Copyright Act of 1976, no
part of this publication may be reproduced or distributed in any form or by any
means, or stored in a data base or retrieval system, without the prior written
permission of the publisher, with the exception that the program listings may
be entered, stored, and executed in a computer system, but they may not be
reproduced for publication.

1234567890 DODO 89876543

ISBN 0-88134-101-0

Judy Ziajka, Acquisitions Editor
David Ushijima, Technical Editor
Fausto Poza, External Technical Reviewer
Ted Gartner, Copy Editor
Paul Butzler, Text Design
Yashi Okita, Cover Design

Unless otherwise noted, all photographs by Richard Cash

Dedication

To my wife Kathe, without whose patience and assistance this book would never have been written.

ACKNOWLEDGEMENTS

I gratefully acknowledge:

Apple Computer, Inc., for supplying software and technical information on the Apple III.

Richard Newman of Diversified Electronics (Keene, NH) for supplying hardware and backup technical assistance.

Tim Gill and Bruce Harris of Quark Engineering (Denver, CO) for supplying copies of Word Juggler and Catalyst.

J. Daniel Rimes of Amdek Corp. (Elk Grove Village, IL) for the loan of a color monitor.

"T.B." for proofreading the original manuscript and providing encouragement and constructive criticism.

Contents

	Introduction	xi
Chapter 1	Introducing the Apple III	1
Chapter 2	Getting Started with the Apple III	19
Chapter 3	Files, Paths, Utilities	39
Chapter 4	The System Configuration Program	59
Chapter 5	Getting Started with Business BASIC	77
Chapter 6	Programming in Business BASIC	95
Chapter 7	Advanced Programming in Business BASIC	125
Chapter 8	Creating Graphics and Sound	165
Appendix A	Business BASIC Quick Reference	187
Appendix B	Business BASIC Error Messages	221
Appendix C	Business BASIC Reserved Words	229
Appendix D	System Error Messages	231
Appendix E	Console Reference	239
Appendix F	Graphics References	247
Appendix G	Printer Reference	253
Appendix H	RS-232C Reference	255
Appendix I	Audio Reference	259
Appendix J	ASCII Character Codes	261
Appendix K	Apple II Emulation	267
	Index	271

Introduction

The book you're holding in your hands is designed to be a *complete* guide to the Apple III personal computer system. The Apple III is a powerful and versatile system that is designed to handle the most involved tasks. For the first time, everything you need to know about setting up and using your Apple III is all in one place. There is no need to continually flip back and forth between numerous manuals.

The first two chapters are an overview of the Apple III and how to make it work. Chapter 1 is a detailed introduction to the system, including a look at the most popular accessories. Chapter 2 explains hooking up the system, getting started, and making backups of your software. It will also be useful to you if you purchase peripherals in the future.

Files are the means by which the Apple III stores and processes data and programs. Chapter 3 explains files, how they're handled, and how to use the Apple III System Utilities. disk.

The Apple III's sophisticated operating system (SOS) is one of the most powerful available in *any* personal computer. Chapter 4 introduces the SOS and goes into detail on how to use the System Configuration Program to customize the SOS to your particular system.

Chapters 5 and 6 are an introduction to Business BASIC, the Apple III's powerful and advanced BASIC interpreter. For people with little or no prior experience with BASIC, Chapter 5 explains the mechanics of using Business BASIC; Chapter 6 eases you into writing your own programs.

If you're an experience BASIC programmer, you can go directly to Chapters 7 and 8. Chapter 7 details the advanced features of Business BASIC that allow you to write professional programs for serious applications. Chapter 8 shows you how to create graphics and music with you Apple III.

The first three appendixes are a comprehensive reference to Business BASIC. Appendix A is a description of every statement and function available in Business BASIC. It's designed to serve as a handy reference once you become familiar with Business BASIC. Appendix B explains Business BASIC's error messages and what to do about them. Appendix C is a full list of Business BASIC's reserved words.

Appendix D covers the Apple III's system error messages, what they mean, and what to do about them. Appendix E is a complete guide to the console, including keyboard codes, cursor and console control keys, screen console codes, and cursor movement options.

Appendix F is a quick reference to creating graphics with your Apple III. It includes a summary of graphic procedures and functions, color codes, and graphics modes.

Appendix G is a summary of the parameters you need to hook up a printer. Appendix H provides detailed information on the Apple III's RS-232-C serial interface. Appendix I is a quick reference to creating music with your Apple III.

The Apple III can also run *Apple II* programs and use many Apple II peripherals. Appendix K details options and limitations of Apple II emulation.

Introducing The Apple III

1

The Apple III personal computer is a sophisticated and versatile system. If you already own an Apple III, we don't have to convince you of this. However, whether you're already the proud owner of an Apple III or are seriously considering buying one, we suggest you read this chapter carefully. Whether the Apple III is your first personal computer or the latest in a series, this chapter is designed to give you a thorough background on the system's features and capabilities.

OVERVIEW OF THE SYSTEM

Figure 1-1 shows a basic Apple III system, including the system unit (the box with the keyboard, disk drive for floppy disks, and electronics) and the Apple Monitor III, a 12-inch video monitor (high-quality television screen).

Figure 1-2 is a picture of what might be considered a "loaded" system. To the basic system shown in Figure 1-1, a second, 5 1/4-inch floppy disk drive, a Profile hard disk (a special type of information storage peripheral that stores the equivalent of 35 floppy disks), a high-resolution RGB (red-green-blue) video display for showing eye-popping color graphics, and a printer have been added. (In this case, the printer is an Epson MX-80 F/T, although nearly any printer will work with the Apple III.)

Your Apple III system probably won't look exactly like either of these,



Figure 1-1. Basic Apple III system



Figure 1-2. Expanded Apple III system

since there are many ways to configure a system. The Apple III also comes with either 128K or 256K bytes (1K equals 1024) of programmable user memory called RAM, for random-access memory. This is where the computer stores programs to be run and the results of its calculations.

There are also four expansion slots inside the case where you plug in accessory circuit boards like controllers for the hard disk drive or a high-speed printer.

No matter how you've customized your system, there will be a number of common features. Of course, there is the Apple III itself, including the built-in keyboard, the floppy disk drive, and the video monitor. In this chapter, you'll get a close look at each of these, some common peripheral equipment, and the software needed to get the system up and running.

THE APPLE III KEYBOARD

Figure 1-3 is a close-up of the Apple III's keyboard. Although it looks like a separate unit, the 74-key keyboard is permanently attached to the main case. If you're an experienced typist, you'll find the Apple III's 61-key main keyboard familiar and comfortable; it's set up similar to a normal typewriter. If you're a touch typist, you'll immediately notice raised dots on the **D** and **K** keys to let you know where they are without having to look at the keyboard.

If you haven't used a computer keyboard before, you'll notice several keys that are new. **ESCAPE** and **CONTROL** are special keys you'll use often in the course of using the system. You will normally press **ESCAPE** followed by another key to initiate a command. Another common way to enter commands called control characters is to hold down the **CONTROL** key while typing another character. The use of **ESCAPE** and **CONTROL** depends on the program you are running.

There are also **OPEN APPLE** and **CLOSED APPLE** keys located on the lower left-hand side of the keyboard. They can be programmed to perform special functions that would normally take many keystrokes.

The four arrow keys on the lower right-hand side of the keyboard are used to control the *cursor*—the solid square that appears on your video display and tells you where you're working on the screen.

The majority of the character keys on the keyboard automatically repeat if they are held down for more than half a second. The arrow keys have two auto-repeat speeds. Touch them lightly and the cursor moves slowly; touch them harder and the cursor moves faster.



Figure 1-3. Close-up of the Apple III keyboard

A full 13-key calculator keyboard is included for applications where you'll be entering a good deal of numeric information. On the upper right-hand side of the keyboard is the **RESET** key that resets the entire system. (This is known as a *warm boot*.) With the Apple III, a two-key, fail-safe system has been added to make sure you don't accidentally reset the system. When you need to reset the system, you have to hold down the **CONTROL** key and press **RESET**.

FLOPPY DISK DRIVE

Immediately above the keyboard on the right side of the Apple III's main case is the built-in, 5 1/4-inch floppy disk drive (shown in Figure 1-4). This drive stores 140K (143,360) bytes of data on a standard miniature disk (the equivalent of about 35 pages of single-spaced, typed text.) This drive is used for loading the software to get the Apple III up and running—a process called *booting* the system. There are several options for increasing your system's disk storage capacity, which will be detailed later in this chapter.

REAR PANEL

If you're like most personal computer users, you'll find you want to expand your system as you go along. The Apple III makes it easy, since the connectors for the most commonly used accessories are built into the rear of the case. Figure 1-5 shows the Apple III's rear panel. Working from left to right, the connectors are

- A connector labeled **FLOPPY DISKS** for add-on, 5 1/4-inch floppy disk drives. As many as three additional drives can be added.

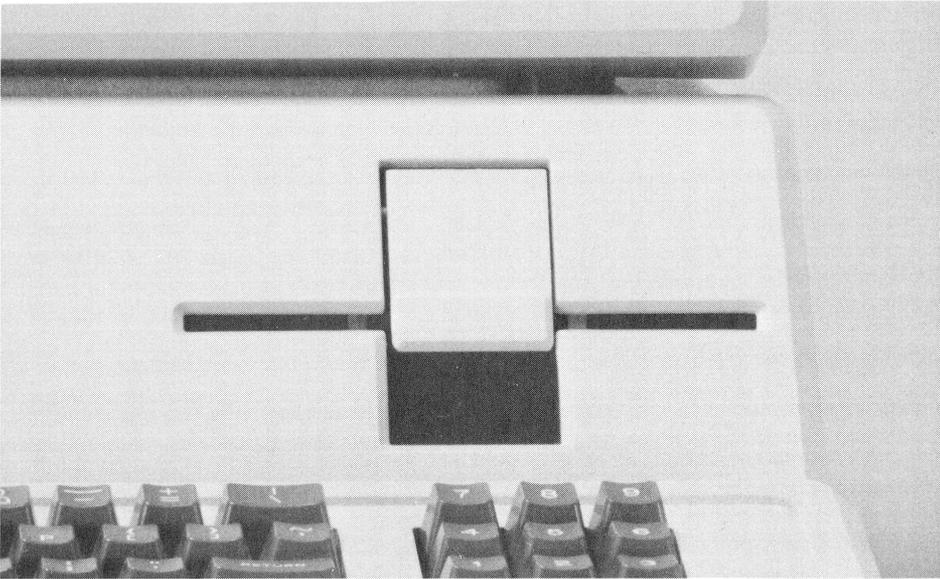


Figure 1-4. Built-in floppy disk drive

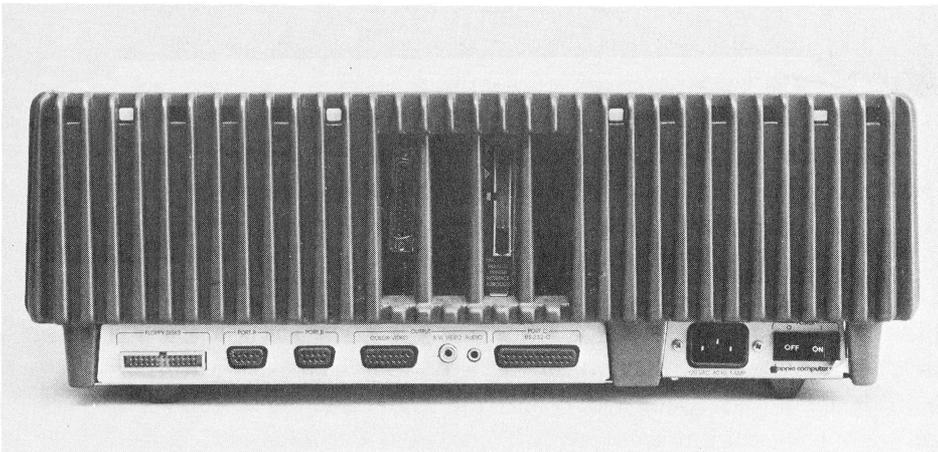


Figure 1-5. Apple III rear panel

- Two connectors labeled PORT A and PORT B for joysticks (for playing games and using special software). The connector marked PORT B is also where an Apple Silentype Thermal Printer is plugged in.

- A connector labeled COLOR VIDEO to which a color video monitor can be connected.
- A connector labeled B/W VIDEO for a black-and-white video display.
- A miniature phone jack labeled AUDIO for connecting an external speaker. (There's also a 2-inch speaker built into the main case.)
- A connector labeled PORT C to the built-in, RS-232-C serial interface. It's used for hooking up a modem (modulator/demodulator) so you can link your computer to other computers over telephone lines. You can also plug serial printers into it.
- A connector marked 120 VAC for connecting the Apple III to the power line.
- The ON/OFF switch for turning the Apple III on and off. The switch is located on the far right.

Notice the four vertical slots in the middle of the rear panel above the connectors. Behind the slots are places to plug in as many as four add-on circuit boards. (In Figure 1-5, two boards can be seen in the slots: a Profile hard disk interface and a parallel interface card for connecting a printer.)

VIDEO DISPLAY

Video monitor, video display, Cathode Ray Tube (CRT)—they're all different terms for what's essentially the same thing—a television screen for reading information (either text or graphics) from the computer. Although you can use a television set as a monitor for the Apple III, it's *not* recommended. A device called an *rf modulator* is required to convert the Apple III's video signal to a signal that a television set can use. The Apple III's high-quality video output requires a high-quality monitor designed specifically for computer use. Most television sets don't have the resolution or circuitry needed to display a full line of 80 characters or high-quality graphics.

Figure 1-6 shows the Apple Monitor III, a video monitor that is capable of displaying high-resolution graphics. It is available in either standard black-and-white or with a green phosphor (the coating on the inside face of the display screen). The green phosphor causes characters to be displayed as light green characters on a dark background and tends to be easier on your eyes. It's highly recommended that you buy the green display if you'll be using your Apple constantly. The Monitor

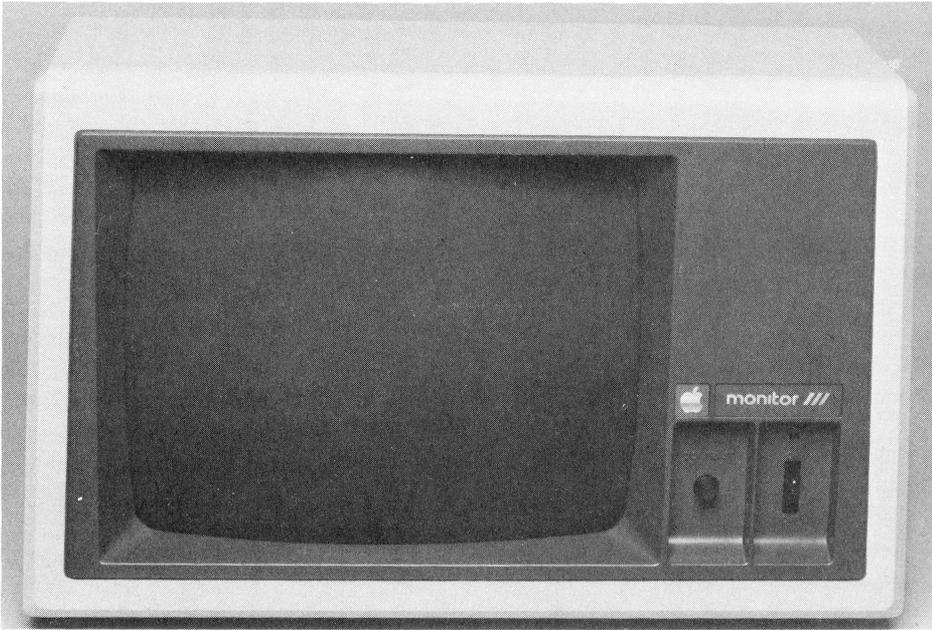


Figure 1-6. Apple Monitor III

III displays a sharp line of 80 characters and the Apple's highest-resolution graphics. If you don't want the Apple Monitor, which is designed to fit nicely on the top of the computer case with the Profile hard disk in between, there are numerous other monitors you can use.

To get the greatest use of the Apple III's outstanding color graphics, you should consider buying a high-resolution color monitor. Although Apple doesn't offer one, many brands are available. There are primarily two types: *composite video* and *RGB*. Composite video monitors are the less expensive of the two and offer high quality at a reasonable price. They can also be used as monitors for video cassette recorders and video games.

RGB (red-green-blue) monitors offer the ultimate in color quality with separate inputs for each of the three primary colors that make up a color television image. They normally can't be used for any other applications. Figure 1-7 shows a typical RGB monitor.

What's on the Screen

The Apple III can display characters and graphics (pictures) in a number of different ways. These displays are called *modes*. In addition,

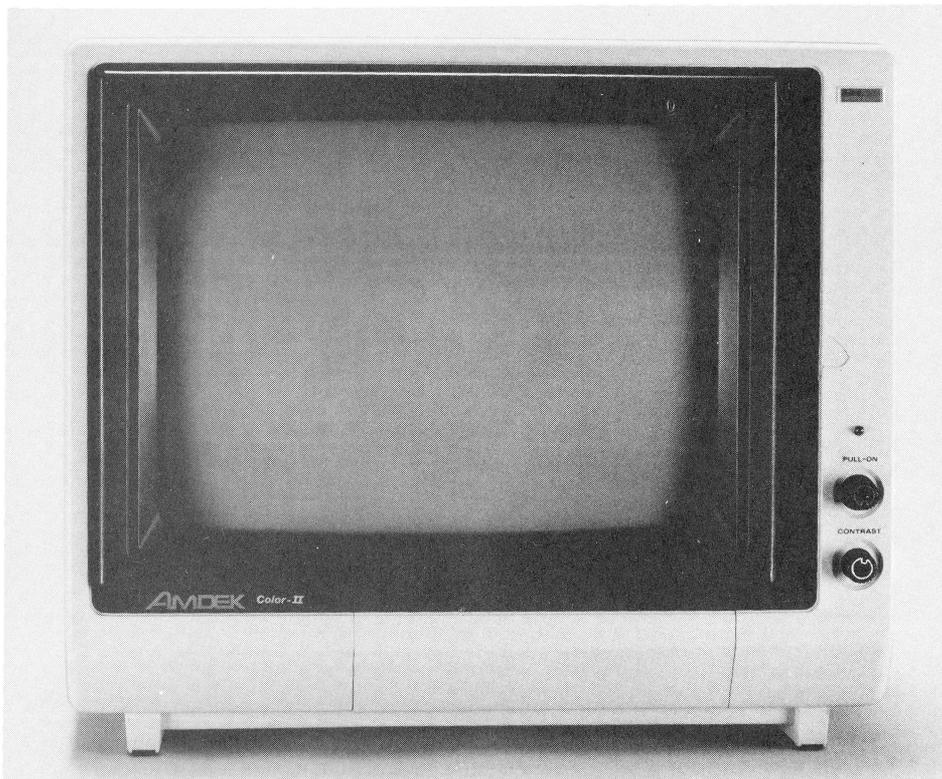


Figure 1-7. RGB color monitor

both upper- and lowercase letters can be displayed without using a special circuit board, making the Apple III ideal for word processing. The three text modes are

- 24 lines of 40 characters each in black-and-white.
- 24 lines of 80 characters each in black-and-white.
- 24 lines of 80 characters each using 16 colors in the foreground and background.

In each text mode, characters can be displayed in either normal or inverse video. Normal is light characters on a dark background, and inverse is dark characters on a light background. Most people find that normal video is the easiest to read and work with.

The Apple III also has four different modes for displaying graphic data (charts, graphs, and so on). Each graphic mode determines the number of pixels (picture elements or individual dots) that are on the

screen at one time. The Apple III's graphics modes have screens that consist of

- 280×192 pixels (53,760 individual points) in black-and-white
- 280×192 pixels (53,760 individual points) in 16 colors
- 560×192 pixels (107,520 individual points) in two colors only
- 140×192 pixels (26,880 individual points) in 16 colors.

INSIDE THE APPLE III

Inside the main case (often called the *system unit*) of the Apple III, lurking behind the keyboard, is the circuitry (the *hardware*) that controls everything the system does. It includes the memory, the microprocessor, the built-in disk drive, and all the wires that allow information to travel from one place to another.

Figure 1-8 shows the Apple III with its “top down.” Admittedly,

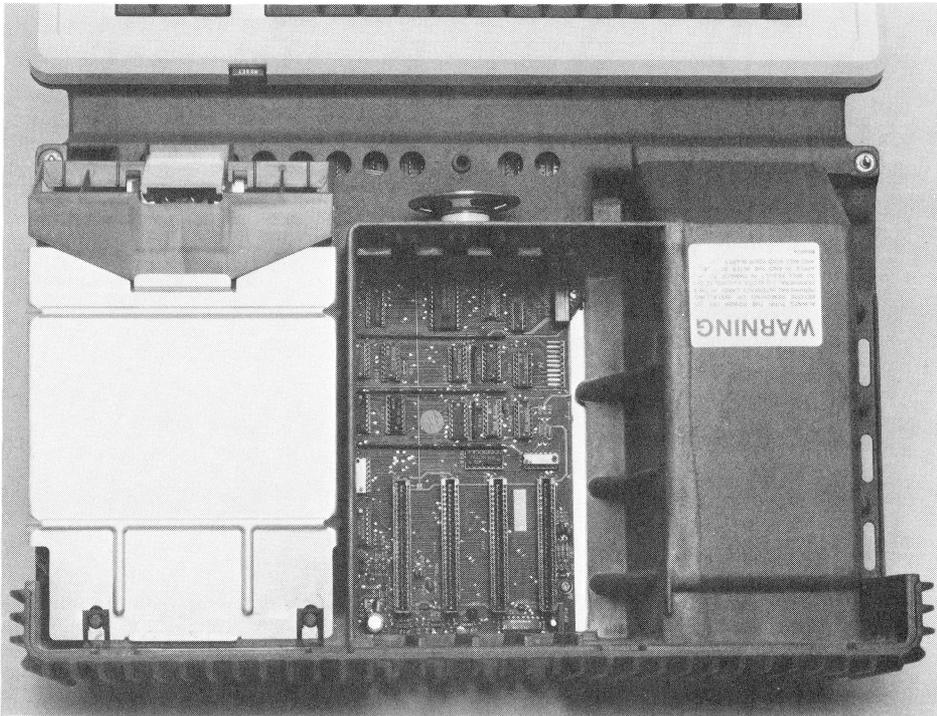


Figure 1-8. Inside the Apple III main case

there isn't much to see—just the floppy disk drive (on the left), the speaker, and the four slots for plug-in circuit boards.

The rest of the Apple III's circuitry is encased in the metal enclosure on the left and underneath the keyboard. The enclosure is there for a good reason. It enables the Apple III to conform to the strict RFI (radio frequency interference) standards set by the Federal Communications Commission. Because microprocessors operate at millions of cycles per second, they emit signals that can be picked up by radios and televisions, causing broadcast interference. The metal shield keeps the signals inside the case so you can listen to a radio or have a television on in the background while you're using your Apple III.

Figure 1-9 shows the circuitry inside the system unit.

WARNING: Don't remove the circuit board as we've done for this photograph—it could void your warranty.

To give you an idea of where the board sits in the Apple III case, notice the four slots on the upper left-hand side of the circuit board. They're the same slots you saw in Figure 1-8.

The circuit board is full of integrated circuits (also called *ICs* or *chips*). In each of the ICs are thousands of microscopic circuits. The "brains" of the Apple III, the 8-bit 6502B microprocessor, is the right-most of the three large chips on the bottom of the circuit board. The chips around it and on the rear part of the circuit board work with the microprocessor to process information and store programs and data.

MEMORY

All computers (and your Apple III is no exception) have two kinds of electronic memory: RAM and ROM.

RAM stands for *random-access memory*, although it's more technically correct to call it user programmable read/write memory. This is where the Apple III temporarily stores programs and data. Every time the AC power is turned off, all the information in RAM is lost. By that time, you'll either have printed out your information or stored it on a floppy disk for later use.

The size of computer memory is measured in *bytes*. Each byte consists of eight bits (binary digits or individual 1s or 0s) and can store one character (a letter, number, punctuation mark, or even a space). The size of the memory is stated in "K" bytes (1024 bytes). Depending on which configuration you purchase, the Apple III comes with either 128K (131,072) or 256K (262,144) bytes of RAM.

The amount of available memory points to another of the Apple's solid advantages. Most 8-bit microprocessors can only address (read from or write to) a maximum of 64K (65,536) bytes of RAM. But the Apple III can handle much more because of its sophisticated circuitry.

What's the advantage? Simply put, the more RAM your computer has, the better. The Apple III's huge RAM space is used by much of the sophisticated software sold for the computer. Not only can large programs do more, but with that much RAM available, they can efficiently handle large amounts of data.

In Figure 1-9, part of the RAM is on the top circuit board. Actually, what you see is a 256K system with 128K on the top board and 128K hidden underneath.

That raised circuit board points out another advantage of the Apple III. If you ever decide to expand your 128K system to 256K, your dealer can easily install the memory expansion without the need to use one of the four expansion slots.

ROM stands for *read-only memory*. ROM is a special memory into which information has been permanently stored. Your Apple III has 4K of ROM. Although that's a comparatively small amount, that's all the Apple III needs. Stored in the ROM is special diagnostic software that tests the system every time it's turned on.

Many small computers store their entire operating system and languages like BASIC in ROM. Although using ROM is a convenient way

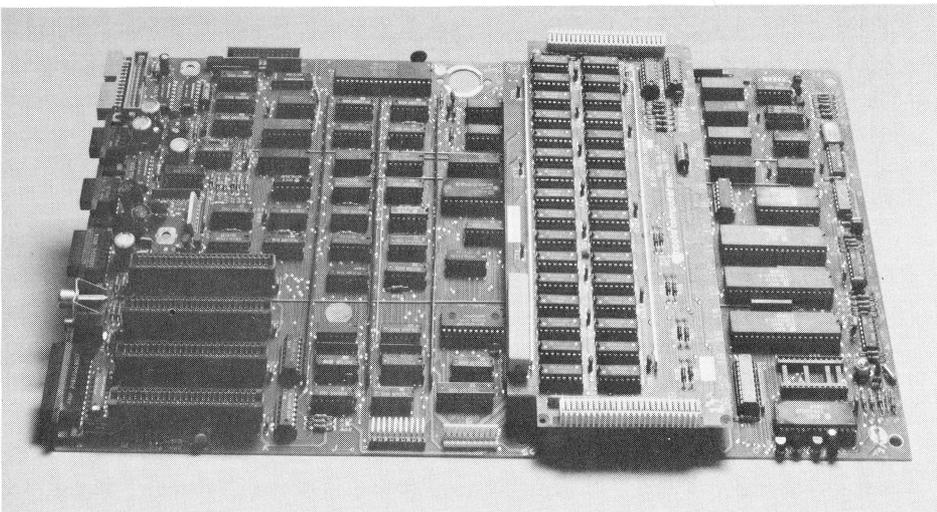


Figure 1-9. Apple III circuit board

to store data, this limits the computer's possibilities. With the Apple III, you load the operating system and whatever language you want to use onto the system from a floppy disk. This only takes a few seconds and makes the Apple III much more versatile. As new and improved versions of the operating system and languages become available, you'll be able to upgrade your system instantly.

SOFTWARE

Now that we've looked at the basic hardware of the Apple III, let's look at *software* (programs). Software is the set of instructions that tells the Apple III what to do. Without software, the Apple III is nothing but a very dumb collection of circuits and wires.

Types of Programs

There are several different types of computer programs. All of them are loaded into the Apple III's RAM from a floppy disk (this is the booting process mentioned earlier).

Applications Programs. These are probably what you think of when you hear the word "program." An applications program is designed to do a specific thing like word processing, accounting, or financial planning. There are hundreds of applications available from Apple and from independent companies for the Apple III. Each differs in the way it works. Our purpose in this book is to explain general operations of the Apple III, not specific applications. Each applications program you buy for your Apple will have specific instructions for using it.

Languages. If you're like many Apple III users, you'll probably be using your computer with pre-written applications programs. But as you experiment more and more with the Apple, you may want to develop customized programs for your own applications.

At the time this book was written, Apple offered three popular programming languages: Apple Business BASIC (Beginner's All-Purpose Symbolic Instruction Code) is the Apple III version of the popular BASIC language; Pascal, an advanced programming language favored by professional programmers; and COBOL (Common Business-Oriented Language), a language used for many business applications programs.

But computers can't directly understand any language. At their most basic level, computers can only understand the combination of 1s and 0s called *machine language*. Two types of special programs called *interpreters* and *compilers* translate programs written in a high-level language

like BASIC or Pascal into machine language. An interpreted language (like BASIC) is translated into machine language each time the program is run. Compiled languages (like COBOL) are first translated into machine language. Since a program written in compiled language is then stored in machine language, it tends to run faster.

Operating Systems. An operating system is an essential program for every computer system. It can be thought of as the “traffic cop” that controls the flow of data into, out of, and through the computer system. An operating system also performs chores like copying data from a disk to memory, getting data from the keyboard, and displaying it on the video monitor.

SOS (Sophisticated Operating System) is the Apple III’s operating system. It has many features that until recently were only available in larger computer systems.

In a later chapter, we’ll explain SOS in detail. For now, you should know that SOS has to be configured (set up) for your specific combination of Apple III and peripherals. SOS communicates with every part of the Apple III system using *device drivers*, special programs that communicate with peripheral devices. Every time you add a new peripheral you’ll have to run something called the System Configuration Program (SCP) to install the proper driver. Your dealer may have already done this for you when you purchased your computer. The configuration process is straightforward, but there’s enough involved to devote an entire chapter (Chapter 3) to it.

ADDING POWER

Although the basic Apple III system is powerful and versatile, there are a number of accessories and peripherals that can be added as your needs require.

We’ve already mentioned video monitors, but both Apple and independent manufacturers offer hardware additions for your Apple.

More Data

As we mentioned earlier, if you purchased an Apple III system with 128K of RAM, you can have it expanded to 256K by your dealer. Whether you need 256K depends on what you’ll be using your system for. Most applications programs don’t need 256K, but as more and more sophisticated software becomes available, you might find that you will need larger and larger amounts of memory. More memory is necessary for programs that do a lot of numeric calculations.



Figure 1-10. Apple Disk III

Floppy Disks

Most people find that they're rather limited by the single floppy disk drive included with the Apple III. Not only do some programs require a second disk drive, but having a second drive makes copying disks much easier. Otherwise, you have to keep swapping disks into and out of the single drive.

Apple offers several alternatives. The Apple Disk III (shown in Figure 1-10), is an add-on, floppy disk drive that's identical to the disk drive built into the Apple. It can read and write exactly the same amount of data. It plugs directly into the back of the computer and doesn't require modification of your operating system. As many as three additional Disk III units can be connected to the Apple III.

The Apple Unifile and Duofile are high-capacity, floppy disk drives. The Unifile contains one disk drive and the Duofile has two. They can

store large amounts of information on a single 5 1/4-inch floppy disk. The disks that fit into the Unifile and Duofile are special and are not interchangeable with the floppy disks that fit into the Apple III's built-in disk drive. Because of the special disks and the special hardware in the drives, each disk can hold 871K bytes of data, as opposed to 140K for the normal disks. (To put that in some perspective, 871K is the equivalent of over 200 pages of single-spaced, typed text.) If you'll be using your Apple III for applications that require a good deal of information storage, like word processing or accounting, the Unifile and Duofile are lower-cost alternatives to the Profile hard disk.

Cassettes

There's no place on the Apple III where you'll be able to hook up a cassette recorder. You don't need one. (The Apple III has a built-in floppy disk drive.) Besides, cassette recorders are far from being the most convenient way to store and retrieve data. If you have used a cassette recorder with a computer, you'll soon see the advantages of using floppy disks.

Profile Hard Disk

The ultimate in information storage for the Apple III is Apple's Profile hard disk drive, shown in Figure 1-11. A *hard disk* is just that—a hard (as opposed to floppy) disk. Rather than a thin piece of plastic

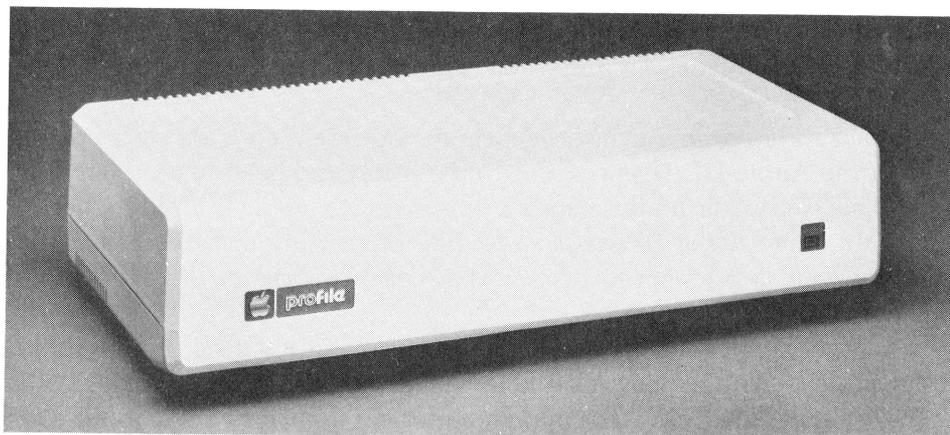


Photo courtesy of Apple Corporation

Figure 1-11. Apple Profile hard disk drive

coated with magnetic material, a hard disk is a magnetically-coated aluminum platter. It's sealed in an air-tight enclosure where the air is continually circulated and filtered. Because of the microscopic tolerances involved, an airborne particle of dust or smoke can destroy the information on the disk.

Hard disks (often called "Winchesters") can't be removed from the drives, but they make up for that disadvantage by storing a *huge* amount of information. A single Profile disk can hold 5 megabytes (a megabyte equals 1,024,000 bytes) of data. That's equivalent to over 1200 pages of single-spaced text. In addition to storing enough data for even the largest application, the Profile disk spins at a much higher speed than a floppy disk and can get information into and out of the computer much faster.

When you look at a hard disk on a cost per byte of storage basis, it becomes a cheap alternative if you're using the Apple III for an application that requires lots of data (for example, writing novels or keeping track of hundreds of customers).

Admittedly, the Profile disk is comparatively expensive, as is any hard disk, but if you need the storage capacity, it can't be surpassed.

Printers

Printers are an important part of any system. It's almost impossible to do without them. The problem is often the choice. There are literally hundreds available, ranging in price from a few hundred dollars to several thousand dollars.

Apple offers three printers for the Apple III: a low-cost thermal printer that requires special paper, a dot-matrix printer, and a more expensive daisy-wheel printer that produces output identical to a high-quality office typewriter. In between are many printers from many manufacturers.

What you need in a printer is largely dependent on what you're doing with the Apple III. If you're word processing and need to send out professional letters or manuscripts, a daisy-wheel printer is almost a necessity. If you're doing financial work like accounting or forecasting, you might want a high-speed printer that can print reams of reports in a short time.

A good compromise is one of the many medium-speed, high-quality dot-matrix printers now available. An example is the Epson printer, shown in Figure 1-12. Like many printers in its price range, it has several modes of operation, which range from fast printing to slower printing with a higher quality image.

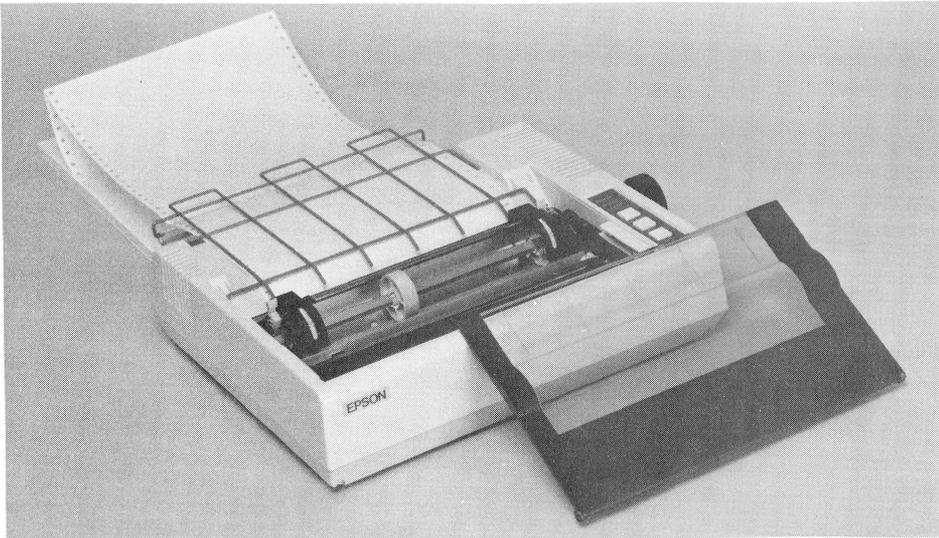


Figure 1-12. Epson MX-80 F/T Printer

Modems

Another very useful peripheral is a *modem* (modulator/demodulator). It allows you to hook up your Apple III to telephone lines to access public databases like CompuServe, Dow Jones News/Retrieval, and The Source, or to swap files with other computer users.

Like so many peripherals, modems come in a variety of “flavors” at a variety of prices. Some are acoustic couplers that have rubber cups into which you place the telephone handset. The Apple III can use many modems designed for the Apple II. They plug directly into the expansion slots inside the computer.

By far the most popular variety of modem is the direct-connect modem, which plugs directly into the phone jack on your wall. Figure 1-13 shows a typical modem set-up.

In order to use a modem, you’ll need special *communications software* (sometimes called a “smart terminal” or “file transfer program”). Apple offers a package called Access III, and many others are on the market. See your local dealer for more information.

Apple II Peripherals

With its special ability to act like an Apple II, the Apple III can also use some of the Apple II peripherals. In particular, since the slots for

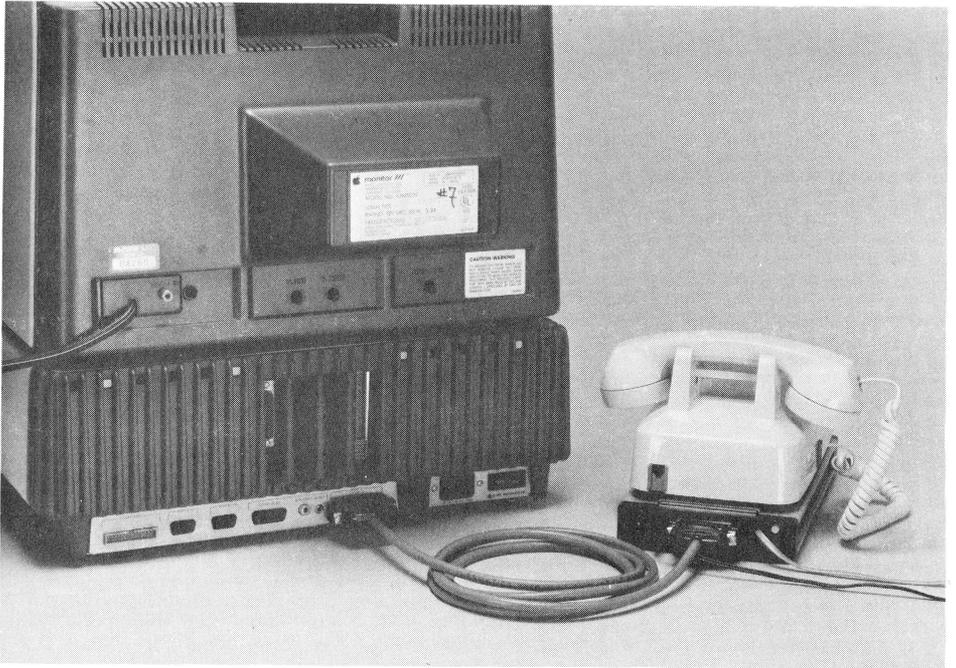


Figure 1-13. Modem with the Apple III

plug-in boards are identical to those in the Apple II, some of the plug-in boards for the Apple II will work in the Apple III.

Note that we said *some*. Because of the hundreds of peripherals and accessories made for the Apple II, it's impossible for us to be specific about which will work and which won't. For instance, you can't use the Apple II Language System (Pascal), although you can adapt most programs to run on the Apple III (as long as you have Apple III Pascal). Also, neither the Apple II Integer BASIC or Applesoft II firmware cards will work. If you're in doubt, the best bet is to see your local Apple dealer.

Getting Started With the Apple III

2

In Chapter 1 we gave you an overview of the Apple III system, its features and capabilities, as well as the hardware, software, and accessories you can buy for it. In this chapter, we'll get down to specifics and the process of setting up and using your Apple III.

First, we'll review getting the system unpacked and set up. (If your system is already set up, skip ahead a few pages to the section of this chapter titled "Using the System.")

UNPACKING THE APPLE _____

Depending on exactly which combination of the Apple III, peripherals, and software you purchased, you'll have quite a few boxes sitting in front of you. In each box Apple includes a list of the box's contents. Make sure everything is included. (We won't give you a list since the exact contents change from time to time.) If anything is missing, contact the dealer you purchased the computer from or fill out the missing parts form enclosed in the box and send it to the address indicated.

SETTING UP THE SYSTEM _____

Before you can use the Apple III, you'll need to set up your system. The exact steps you'll need to perform depend upon the system you've purchased; however, the following sections review the usual processes for setting up a system.

Hooking Up the Apple III

All of the connections you'll have to make are located on the rear panel of the Apple III. Place the Apple on a desk or a table with the rear of the computer facing you.

Make sure the ON/OFF switch (located on the right side of the case) is in the OFF position, and plug the AC power cord into the socket located next to the switch (see Figure 2-1). Plug the other end of the cord into the nearest three-wire grounded wall outlet. Make sure that the socket is grounded. A non-grounded outlet is not only dangerous, but can also cause a buildup of static electricity that can disrupt the operation of your Apple III.

Next, connect the video monitor, as shown in Figure 2-2. If you're using a monochromatic monitor (black-and-white, green-and-white, or amber-and-white), you'll be able to use the cable included with the Apple III. One end goes into the socket marked B/W VIDEO on the rear panel of the Apple III and the other plugs into the monitor.

If you're using a color monitor, you'll use the socket on the Apple III marked COLOR VIDEO. If you buy a color monitor, make sure the dealer provides you with the proper cable for hooking up the monitor.

That's all that's required to hook up the basic system. However, if you're like most Apple III owners, you also purchased a few additional devices known as *peripherals*. Let's look at how to hook them up.

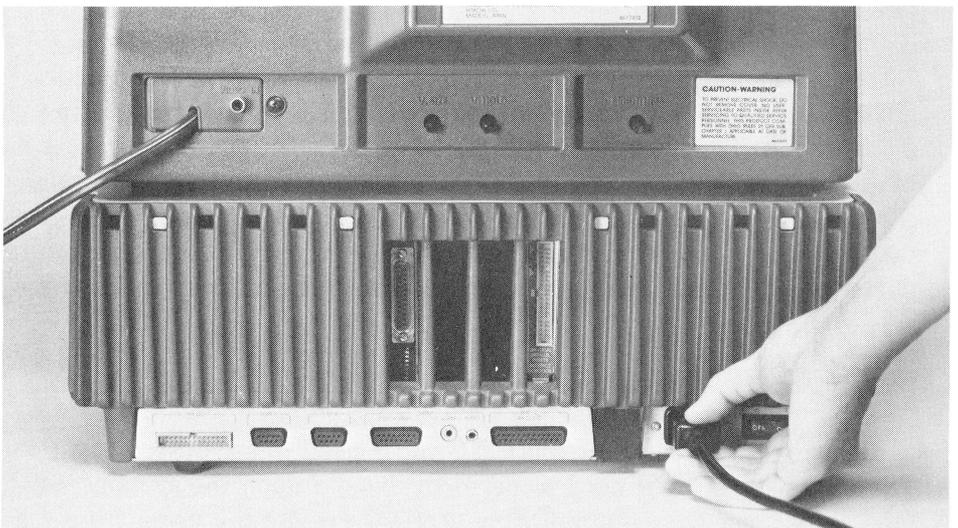


Figure 2-1. AC connector at the rear of the Apple III

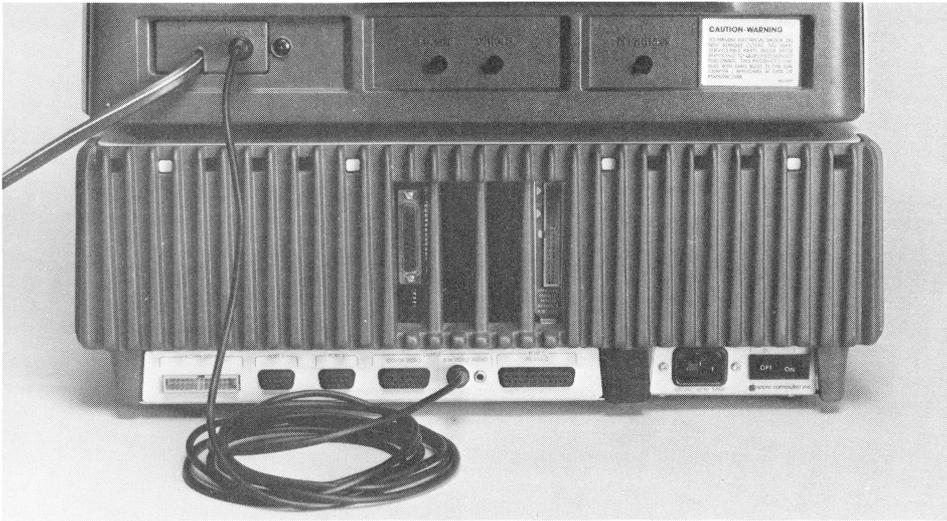


Figure 2-2. Connecting the video monitor

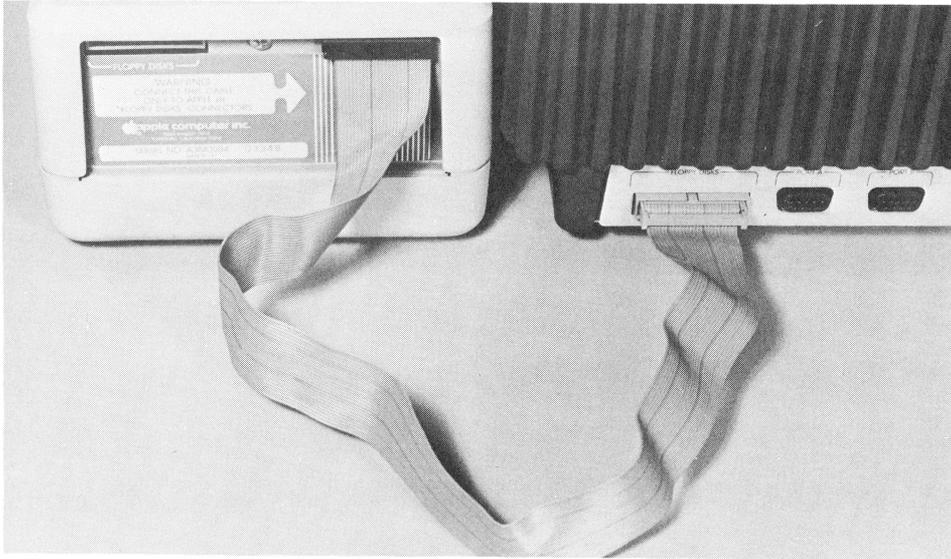


Figure 2-3. Connecting the Apple Disk III

Installing an Apple Disk III

As shown in Figure 2-3, the Apple Disk III plugs into the leftmost socket, marked FLOPPY DISKS. This socket also supplies electrical power to the drive so that you don't have to plug it into a wall outlet.

Add-on Boards

Several of the Apple III's peripherals require you to plug a circuit board into one of the expansion slots located inside the Apple III's case. To gain access to these slots, loosen the two screws on either side of the case underneath the keyboard. (See Figure 2-4.) Then place your fingers under the main case and pull upward as shown in Figure 2-5. The expansion slots (shown in Figure 2-6) are located in the center of the chassis.

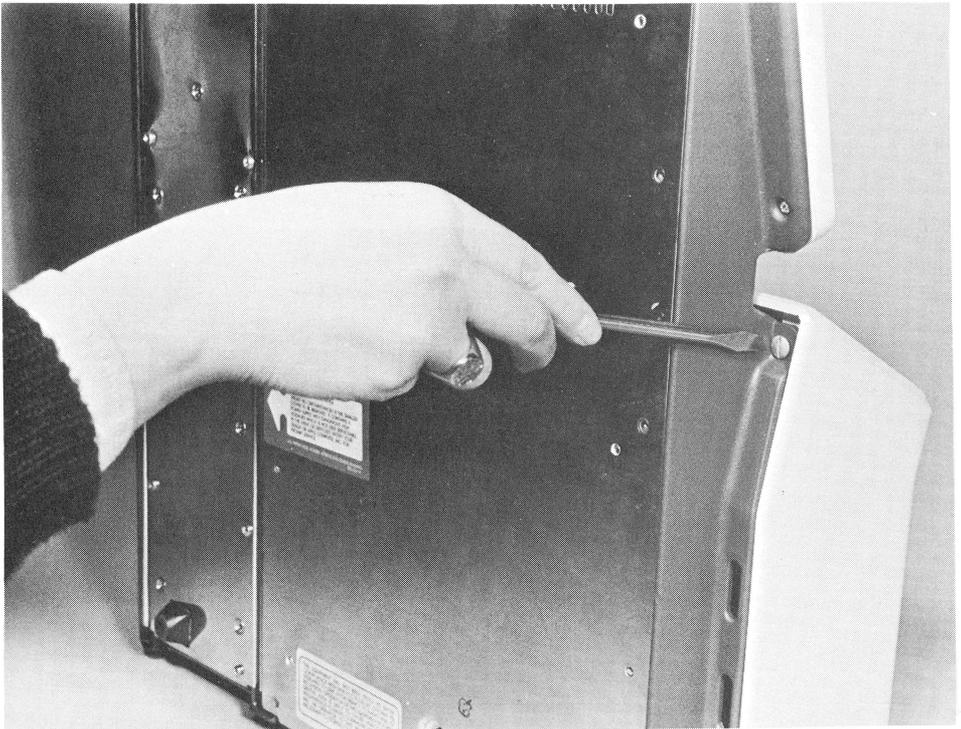


Figure 2-4. Removing the cover screws

Installing the Profile Hard Disk

If you purchased the Profile hard disk drive, we'll explain the intricacies of using the drive later in this book. But in the meantime, let's install it.



Figure 2-5. Removing the cover

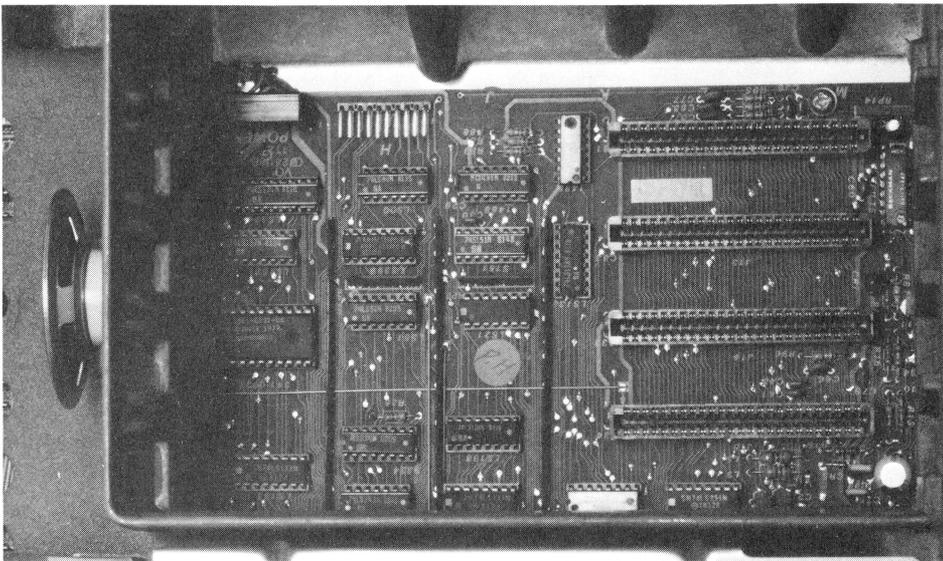


Figure 2-6. Expansion slots

NOTE: The Profile is a *very* fragile piece of equipment. You probably already realized this when you saw the box it was packed in with its numerous precautions. Because of the microscopic tolerances involved,

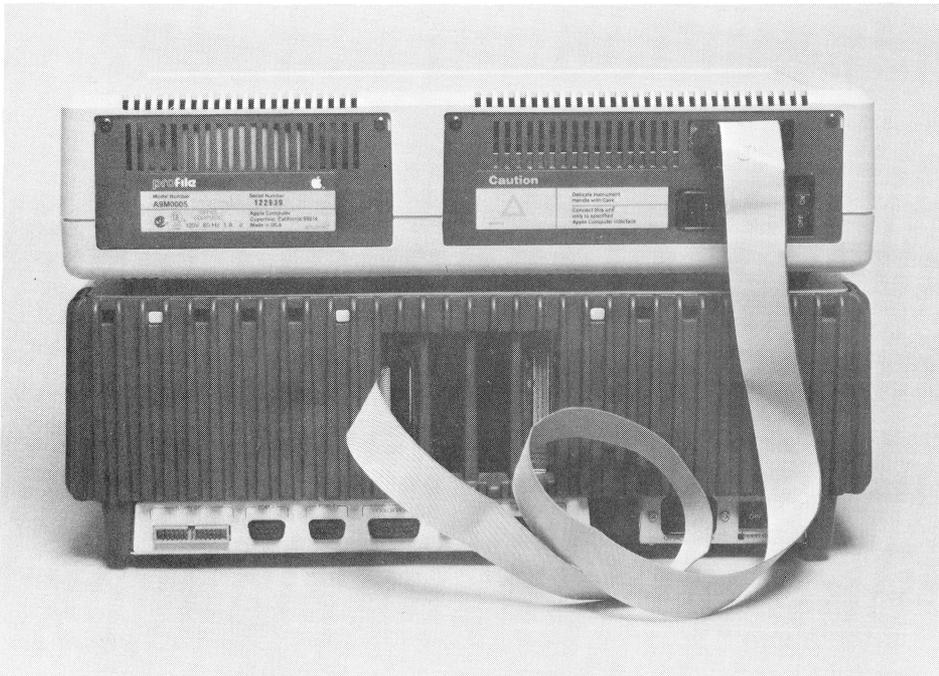


Figure 2-9. Connecting the Profile hard disk drive

depends on your system and what other peripherals you'll be using. See Apple's *Profile Owner's Manual* for specifics on which slot to use for your particular system.

Figure 2-9 shows how the Profile is hooked up. A cable plugs into the connectors on the interface card and the back of the Profile. The Profile also requires its own AC power (a cord is included).

In order to use the Profile, you'll have to install a program called a *device driver* onto your operating system disk. We'll explain the details in Chapter 4.

Installing a Printer

How you connect a printer to the Apple III depends upon the type of printer that you have. The Apple Silentype thermal printer attaches to the socket marked PORT B at the rear of the Apple III, while the Apple daisy-wheel printer attaches to the socket marked RS-232-C.

Many popular parallel interface printers, including Apple's dot-matrix printers, require you to install the Apple Universal Parallel

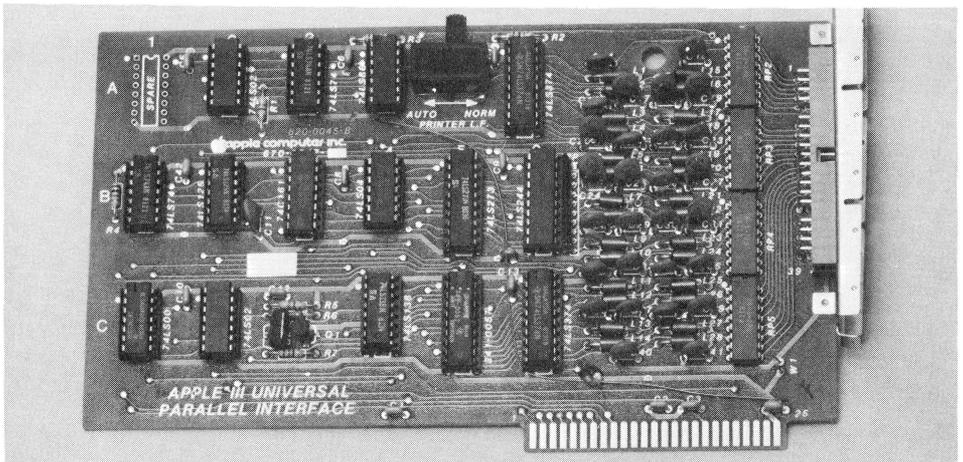


Figure 2-10. Apple Universal Parallel Interface Card

Interface Card (UPIC), shown in Figure 2-10. This card plugs into one of the Apple III's expansion slots and provides a standard way of connecting a parallel interface printer to the Apple III.

Since a number of different printers can be connected to the UPIC, you will need to refer to the UPIC manual in order to properly set up the board. In addition, you will need to install the proper device driver on your operating system disk for the printer you are using (a subject we will cover in Chapter 4).

Installing a Modem

If you'll be using your Apple III to communicate with "information utilities," like The Source or CompuServe, or with other computers through the telephone lines, you probably purchased a modem. Although some modems are designed to plug directly into one of the Apple III's expansion slots, many users buy modems that plug into the Apple III's RS-232-C connector.

USING THE SYSTEM

Before you turn on your system, make sure that you've connected everything correctly. When you're sure that all the cables are hooked up to the right places, give the connectors an extra push and wiggle to make sure they're securely connected. Before we explain all the specific

details of using the Apple III and its software, you should familiarize yourself with the basics of using the system. That's easy to do with the System Demonstration disk packed with your computer. Open the drive, slide the disk in with the label up and toward the front of the computer, close the door, and turn the ON/OFF switch (located on the left rear panel) to the ON position.

The disk drive will whir and click for a few seconds and several messages will appear on the screen. Shortly thereafter, a series of graphics will appear on the monitor. They're designed to show you a few of the capabilities of the system.

If you don't do anything, the demonstration will just go on and on. However, after the graphics have finished, you'll see a list of demonstration programs appear on the screen. This list is called a *menu* because it gives you a list of choices. If you see one that interests you, press the key of the number corresponding to the demonstration. If you do nothing, the graphics will start up again. If, at any point, you want to see the list of choices, press **ESCAPE** and the menu will appear on the screen.

Use the System Demonstration program until you've explored all its possibilities. When you're ready, select option 5 and you will exit from the demonstration.

THE CONSOLE

As you use programs designed for the Apple III, you will run across the term *console* quite a few times. The console is the keyboard and the video screen (no matter what type you're using—black-and-white or color).

Just like the dashboard of a car (also called a console), you'll use the Apple III console to *control* the computer with the keyboard and *monitor* what is happening inside the Apple III by reading the messages that appear on the video display.

By far the best way to see how the console works is to use it. The Console Demonstration program is a good way to get used to the Apple III keyboard.

THE APPLE III KEYBOARD

The Apple III keyboard (shown in Figure 2-11) closely resembles a keyboard of a standard typewriter. For now, use the keyboard like you would use a typewriter, and don't worry about the special keys.



Figure 2-11. Apple III keyboard

When compared to a typewriter, there are several differences in the way the Apple III handles text. Besides the obvious fact that text appears on a screen rather than on a piece of paper, you don't have to worry about pressing RETURN as you approach the end of the line. The Apple III automatically wraps text around to the next line. Although that sometimes results in a word being split in the middle, it makes no difference to the computer. It still "sees" each word individually. (If you'll be using one of the many word processing programs designed for the Apple III, like Apple Writer III, you'll see the *word wrap* feature, where entire words are wrapped around to the next line so they're not split.)

Correcting Errors

Correcting an error on the Apple III is different from correcting an error on a typewriter. You probably noticed that there isn't a backspace key. You'll use the LEFT ARROW key (←) on the lower right. Type in a few words and move the cursor (the solid square that indicates where you're typing) back a few spaces. You'll notice an important fact: the arrow keys *don't change* what they move over. You can back the cursor up anywhere in the line and type over an error to correct it. This feature is called *typeover*.

Note that if you press RETURN when the cursor is in the middle of a line, everything to the right of the cursor is ignored by the Apple III. For instance, if you're using Business BASIC and you type in

```
)PRINT "The Apple III is an advanced computer."
```

when you press RETURN, you'll see

```
The Apple III is an advanced computer.
```

However, if you use the `LEFT ARROW` key to move the cursor back into the middle of “advanced” and then press `RETURN`, you’ll see

```
The Apple III is an adva
```

Everything to the right of the cursor is ignored.

NOTE: If you’re using Apple III Pascal, there are some important exceptions to the way the cursor works. See Apple’s *Pascal Manual* for details.

Deleting a Whole Line

The Apple III also allows you to erase an entire line (or several lines) if you decide you want to retype. It’s done by holding down the `CONTROL` key and pressing `x`. To see how it works, type the following line and then press `CONTROL` and `x`:

```
The Apple III
```

As soon as you’ve done so, the line will look like this:

```
The Apple III\
```

Even though the line of text is still on the screen, the backslash indicates that it’s been ignored by the Apple III. The cursor moves down to the next line and you can start over again.

Typeahead Buffer

A *buffer* is an area in memory where information is temporarily stored until the computer is ready to use it. If you’re a lightning-fast typist, you can sometimes get ahead of the computer. With many computer systems, you actually lose what you type when the computer is busy doing something else. This is not the case with the Apple III. While the computer is busy, everything you type is stored in the typeahead buffer and fed to the microprocessor as soon as the Apple III is ready. Although that might not sound like a big feature right now, you’ll find it very handy in the future.

APPLE III SYSTEM DISKS

In Chapter 3, we’ll take a long look at the Apple III’s Sophisticated Operating System (SOS). As you’ll soon see, SOS is very powerful and

gives you access to features that until recently have only been available with large computer systems.

As we mentioned briefly in Chapter 1, in order for the system to work you must have the SOS set up (configured) for your particular system. The SOS communicates with individual parts of the system using device drivers—special programs that control the major components. A *system disk* is one that contains the operating system program and the proper device drivers for your particular system.

BOOTING THE SYSTEM

“Boot the system” is one of the most commonly heard terms in the world of computers.

The origins of the term come from a small program that actually starts the computer. It essentially “pulls the system up by its bootstraps.” When you “boot” your Apple III (or any other computer for that matter), all you’re doing is loading a program into memory to get the hardware up and running. In the case of the Apple III, it’s done by inserting a system disk into the built-in disk drive and turning on the power.

DISK CARE

Like the other parts of your computer system, floppy disks are fragile and should be handled with care.

Floppy disks are thin pieces of plastic coated with the same type of magnetic material used on recording tapes. But the similarity ends there. If you inadvertently damage a small part of a music cassette, you can still play the music. However, a floppy disk stores data as hundreds of thousands of tiny areas that are either magnetically charged or not charged, corresponding to the 1s and 0s of the binary numbering system. In most cases, the program is ruined if even one of these bits is damaged. Therefore, there are a few rules you should remember about handling floppy disks.

Environment

Although floppy disks are enclosed in a protective plastic jacket to keep the magnetic surfaces from being easily damaged, you should be very careful to make sure you don’t let your fingers touch the magnetic

areas that show through the protective jacket. Your fingers leave microscopic oil residues that can destroy the magnetic surface.

Also make sure that you replace a disk in its protective envelope when it's not being used. And store your disk in a dust-free, smoke-free environment. Smoke and dust particles can scratch the surface of disks if they get under the surface of the plastic wrapper. (Many dealers sell plastic cases that protect disks from such environmental hazards.)

Finally, make sure your working environment is clean. A computer coated with dust is asking for trouble, since the dust will surely get on the disks. And if you must have your cup of coffee while working on the Apple III, keep it on a separate table—away from the computer and any disks.

Handling

Floppy disks get their name from the fact that they're flexible. But they still should be handled carefully. (If you treat them like they're pieces of glass, you're headed in the right direction.) Remember that they're *coated* with magnetic material. If you flex them or abuse them too much, the coating will flake off, and your data will go into oblivion.

Never force a disk into a disk drive. If it does not go in easily, there's a problem. (A common problem occurs when you attempt to force a disk into a slot already containing another disk.)

Temperature

Temperature is another important consideration. Floppy disks are most comfortable at the same temperatures at which people are comfortable. High temperatures or high humidity can damage them. One of the greatest advantages of today's personal computers, like the Apple III, is that they don't require special climate-controlled rooms like the behemoths of old. But if you're using your computer in a region where high temperatures or high humidity are common, you should buy an air conditioner or dehumidifier. It will make both you and your computer very happy.

Magnetic Fields

Because a magnetic field is used to record the information on your floppy disks, any stray magnetic field of sufficient strength can alter or destroy the data on a floppy disk. You might be surprised to find that

sources of magnetism abound in every office and home. In fact, just about everything that uses electricity produces some sort of magnetic field. It's better to be safe than sorry. Keep your floppy disks away from television sets, telephones (the bell uses a magnet), fans, electric typewriters, and stereo speakers (many of which use *huge* magnets).

MAKING BACKUPS

It's good standard operating procedure to make copies of your software and store the originals away in a safe place. (Some people even use a safe-deposit box.) In fact, the best method is to keep two sets of originals by making backups of the "master" disks, labeling your new disks "working masters," and making backups of them (as shown in Figure 2-12). That way, if something unexpected happens (disk drives have been known to make occasional light lunches of floppy disks), you won't be left without a copy of your program.

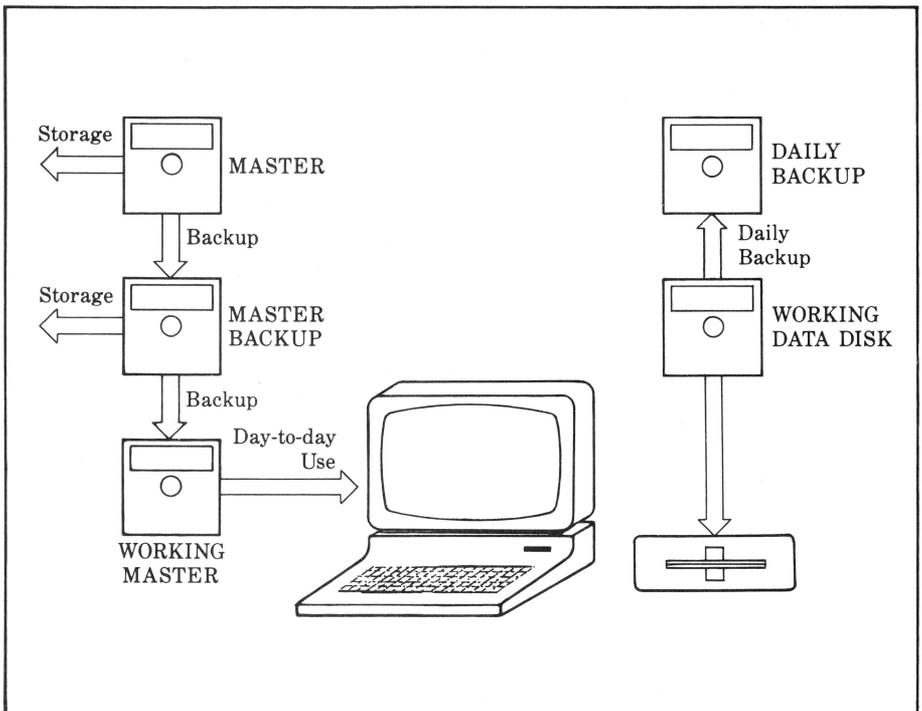


Figure 2-12. Making backup disks

If you're using your Apple III every day, you should make a *daily* backup of your data (as shown in Figure 2-12). Although it looks complicated, it only takes a couple of minutes to copy your work at the end of each day.

You should make backups of the following diskettes that came with your Apple III (disks marked with "*" are optional, depending on which system you purchased):

- System Utilities
- System Utilities Data
- Apple II Emulation
- Apple Business BASIC
- Pascal 1*
- Pascal 3*
- Profile Driver and System Utilities Software*
- Universal Parallel Interface Card Driver Software*

NOTE: Although most of the disks that came with your Apple III can easily be copied, such is not the case with many of the applications programs you'll buy. Many of them are "copy-protected" to prevent people from making copies of very expensive software and distributing them. However, most companies that copy-protect their software make low-cost replacement disks available if the originals are inadvertently ruined.

What You'll Need

If you haven't done so already, purchase at least one box of 5 1/4-inch floppy disks. Ask your dealer for soft sectored, single-sided, single-density disks. (If you have the Unifile or Duofile, you'll need disks specifically designed for these drives. Since they'll *only* fit in these drives, they should be clearly marked "Unifile/Duofile." If in doubt, ask your dealer.) For the standard drives, there's no need to pay the extra money for double-sided or double-density disks. However, buy the best disks you can afford. Although most bargain brands are okay, you should trust your valuable data only to a top-quality disk.

Write-Protection

As you slide a disk into the Apple III's disk drives, you'll notice that there's a square notch cut out from the left side. This is the *write-protect*

notch. When it's covered, the Apple III won't let you erase data that's on the disk or put new data on it. Your master disks should always be write-protected. (Apple master disks come that way.) You can write-protect disks by covering the notches with the peel-off tabs that come with most disks. If you try to copy onto a write-protected disk, the Apple III will tell you (on the screen). Before removing a write-protect tab, make sure you're using the right disk. Once you erase a disk by writing new information onto it, there's no way to bring it back. (That's why you should always make backups.)

Device and Volume Names

Many of the commands (like the COPY command which we will use in a moment) will ask you for the name of the device you will be using. In the case of the COPY command, the *device name* indicates where the data can be found or where the data should be stored.

Device names always start with a "." and are easy to remember. Here are a few:

.CONSOLE	The console (video display/keyboard)
.D1	The built-in disk drive
.D2, .D3, or .D4	Additional disk drives you've added
.PROFILE	The Profile hard disk
.PRINTER	A printer
.PARALLEL	The parallel interface card
.RS232	The RS-232 interface

In addition, when you use storage devices like floppy disk drives, you may use what is called a *volume name* to refer to a particular disk. For instance, the name of the System Utilities disk is /UTILITIES.

Volume names must begin with a slash followed by a letter and can consist of up to 14 additional letters, numbers, or periods; no other punctuation is allowed.

Here are a few legal names:

```
/UTIL.BACKUP
/FORMLETTERS
/MEMOS-1984
```

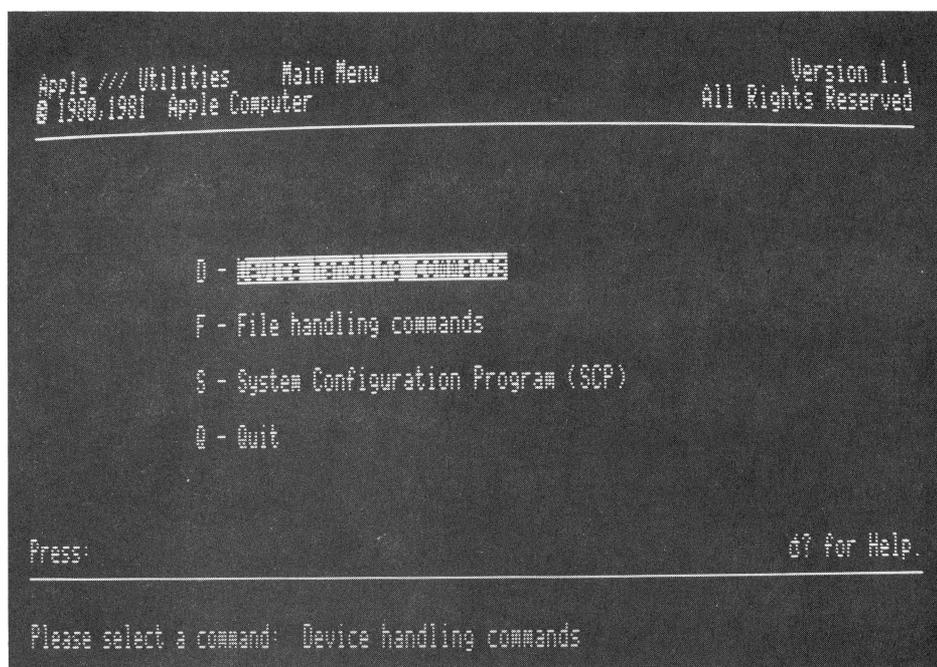


Figure 2-13. Utilities menu

The Backup Process

To show you how the backup process is done, let's back up the System Utilities disk. It's one you'll be using extensively.

If you have a second Disk III drive (or a Unifile or Duofile), backing up disks will be easy and quick. If you'll only be using the built-in drive, it will still be easy, but not as quick. (You'll have to swap disks several times during the backup process.) If you have a second disk drive and haven't connected it yet, do so now.

Make sure the computer is turned off. Place the System Utilities disk into the built-in drive and a blank disk into the second drive (if you have one). Otherwise, keep the blank disk ready.

Turn the Apple III on. In a few seconds, you should see the menu shown in Figure 2-13.

In many of the menus you'll be seeing as you use your Apple III, the highlighted choice on the menu is the one that you'll use most often. We don't want file handling commands this time—we're going to be using the device handling commands. In Apple III parlance, a device is any of the individual parts of the system the operating system communicates

with (video display, keyboard, Profile, Disk III and so on). We want to copy the contents of a whole device (the built-in disk) onto another device (either the same or another disk drive). Press **D** and you'll see the menu shown in Figure 2-14.

This time, the highlighted "Copy one volume onto another" is the right choice, so press **C**.

You should see the following message at the bottom of the screen:

```
Copy the Volume:
[.D2          ]
To the Volume:

With the New Volume Name:
```

The volume name in the brackets is the name of the volume you'll be copying *from*. Since we put the original disk in the built-in drive (which is called .D1), we'll have to change it by typing .D1 and pressing RETURN. (If you make a mistake, use the arrow keys to move the cursor back and type in the correct character. If you get hopelessly mixed up, press ESCAPE to get you out of the backup program and back into the device handling commands menu.)

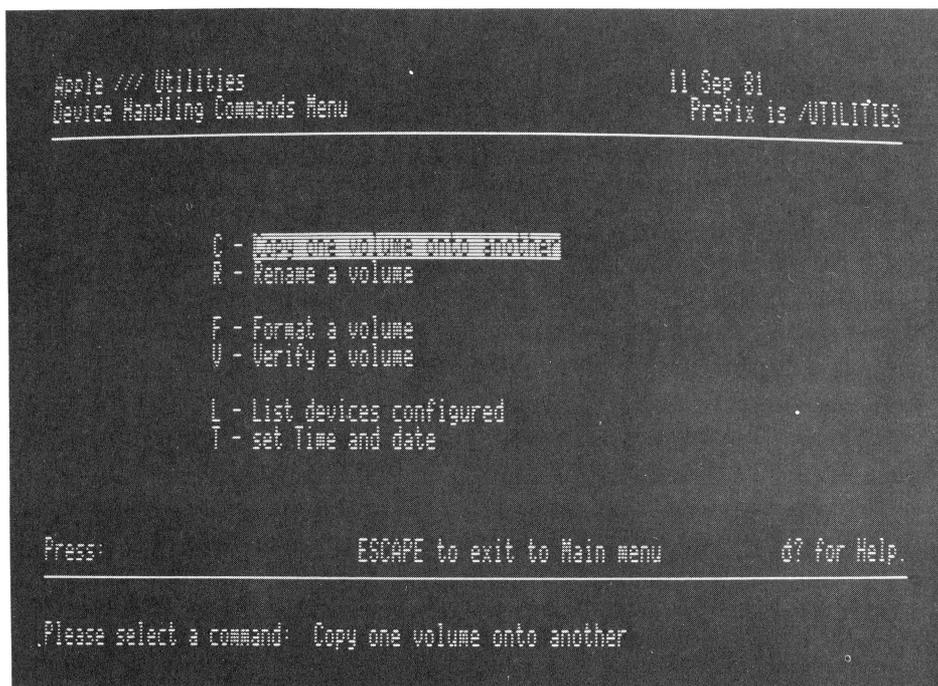


Figure 2-14. Device handling commands menu

Notice that as soon as you entered `.D1`, the computer automatically filled in `.D2` as the volume to copy to. If you have two drives, you're all set. If you don't, enter `.D1` as the name of the volume to copy to and press RETURN.

The Apple III assumes that you want your backup to have the same volume name as the original, so it fills the name UTILITIES into the space provided for the new volume name. If, for some reason, you want to change the name, just type the new name over the existing name and press RETURN. Otherwise just press RETURN.

With two disk drives, that's all there is to it. The drives will whirl and messages to let you know how the backup operation is proceeding will appear on the screen.

With one disk drive, assuming you've typed `.D1` as both the origin and destination drive, the system will prompt you (in a box at the top of the screen) when to insert the *source* volume (the original disk) and the *destination* volume (the blank disk you're copying to). Be forewarned that you'll have to make a number of swaps before the process is finished.

The Apple III first *formats* the disk. This process electrically divides the disk into sections and checks to make sure that it isn't defective. As soon as the disk is formatted, the copying process starts. Depending on the amount of data on the disk, it can take a few seconds or a couple of minutes to completely copy it. A message appears on the screen when the Apple III is done making the backup. At this point, you should remove both disks, file the original in a safe place, and label your copy.

FORMATTING BLANK DISKS

The backup process automatically formats a blank disk before copying all the files from the original disk onto the new disk. However, for most applications, you'll need to have a number of blank, formatted disks on hand for storing programs and data. You can use the device handling commands on the System Utilities disk to format the rest of the blank disks you have.

After selecting the device handling commands, press F to format a blank disk. The following message appears on the bottom of the screen:

```
Format the medium of the volume :
[.D2                               ]
with the new volume name :
```

Once again, the Apple III assumes you have a second disk drive (`.D2`) connected to the system and that you've placed the blank disk in it. If you don't have a second disk drive, type `.D1` in the space. Then press

RETURN and the cursor will move to the field labeled "with the new volume name."

Suddenly a name appears on your screen. The Apple III will automatically name the disk you're formatting BLANK followed by a number. Since at this point you don't know what you'll be using the disk for, the best bet is to use the name the Apple III assigns to the disk. Press RETURN and the formatting process will proceed. (If you want to give it another name, type it into the space provided.)

NOTE: Once again, if you don't have a second disk drive, the system will instruct you to swap disks a number of times. Also, if you've inadvertently attempted to format a disk that already has information on it, the system will ask you if it's okay to destroy what's there. (The formatting process erases all data on a disk.)

A "formatting complete" message will appear on the screen. Press ESCAPE to get back to the Device Handling Command Menu. Finally, make sure to note on the disk's label that it has been formatted. This eliminates duplication of effort later on.

IF YOU HAVE PROBLEMS

The Apple III utilities such as backup and format have been designed to be easy to use and to lead you step-by-step through the process. If you're doing something incorrectly or there are other problems, you'll generally see a message explaining what the problem is. If you don't understand the error message or still don't know what to do next, see the error messages in Appendix D or Chapter 3, where we'll give you a deeper explanation of using the System Utilities disk.

Files, Paths, And Utilities

3

Every computer, from your Apple III to the largest mainframe, operates on *information*. In the world of computers, information is known as *data*. Computers *retrieve* data from storage (a floppy or hard disk), *operate* (perform calculations) on it, and *store* information back into memory or on the disk.

Although data is the basis of our “information society,” it’s nearly impossible to use that information unless it’s organized—put in some type of logical order.

A file is an organized collection of data. Files, the electronic equivalent of the file cabinets and folders you find in every office, are where your Apple III stores data. That data can be programs, letters, mailing lists, and so on.

In this chapter, we’ll take a look at how you can organize, find, and store data in your computer.

TYPES OF FILES

The office filing cabinet is a good analogy to keep in mind when thinking about how your computer stores and uses information. However, unlike the files in the filing cabinet, there are several different types of files. The type of file depends upon where the actual data in the file is stored.

Permanent vs. Temporary Files

Any file that’s stored on a floppy or hard disk is considered a permanent file. When you turn the computer off, the file is still there on the

disk. A temporary file is one that resides in RAM, the electronic memory of the Apple III. When the power switch is turned off, the file is gone forever.

Block and Character Devices and Files

Simply put, a *device* is hardware that the microprocessor communicates with. Even if you have a “stripped down” Apple III, you have two devices right in the case. One is the built-in floppy disk drive and the other is the console—the keyboard and video display. Other devices are printers, modems, and the Profile hard disk drive.

There are two types of devices: *block devices* and *character devices*. A block device stores data in blocks of 512 bytes. (Each byte represents a number, character, or graphics symbol.) Disk drives are examples of block devices.

A character device sends or receives data one character at a time. The screen, keyboard, and RS-232-C serial port are all examples of character devices.

Only block devices are capable of storing files. Although character devices are considered files for the purposes of input and output, they are not capable of storing files.

FILENAME

In the Apple III, each file has a unique identifier called a *filename*. The rules for an Apple III filename are as follows:

1. A filename can contain up to 15 characters and must consist of letters, numbers, and periods (no other punctuation is allowed).
2. A filename must begin with a letter.

Here are a few legal names:

ARTICL.12.02.82

LETTR.TO.JIM

DATA3.00004

When naming a file it makes no difference whether you type in upper- or lowercase letters—the Apple III automatically converts all lowercase letters to uppercase ones.

The following names are incorrect:

12.02.82.ARTICL (Begins with a number)

LETTR TO JIM (Contains spaces)

DATA3:00004 (Contains punctuation other than a period)

FILE DIRECTORIES

You have to have some way to keep track of all the information you've stored on a floppy or hard disk. In an office file cabinet, it's a relatively simple matter of using individual file folders, each with a name or category on the tag.

Much the same is done with your computer files. A disk can contain many files. If you're using a hard disk, there can be hundreds of files stored there. Keeping track of them is done through the use of a *directory*—essentially, a list of files stored on the disk. (Note that the term *catalog* is often used interchangeably with directory in personal computer jargon.) As you'll see later, you use the CATALOG command to get a list of BASIC programs.

A directory is itself a file. If we go back to the file cabinet analogy, you can think of the directory as a folder that contains a list of all the other files in the cabinet (see Figure 3-1). If you want to know what files are in the cabinet, all you have to do is pull out the directory file.

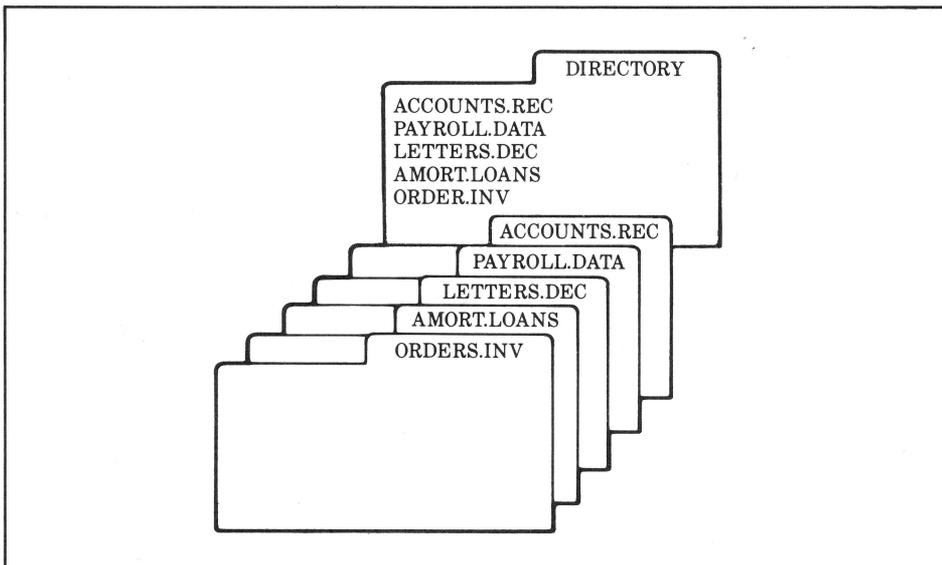


Figure 3-1. The directory file in a file cabinet

The Apple III has a program called the *Filer* that gets the directory from a disk for you to look through. It also allows you to do other things like copy, delete, and rename files.

Subdirectories

A directory can contain a list of files in which some of them are themselves directory files. These second-level directory files are known as *subdirectories*. You'll be using subdirectories a great deal, especially if you have a Profile hard disk.

Subdirectories allow you to organize large amounts of information in a logical manner. A subdirectory can contain a list of related files that do not appear when you look at the normal directory.

Let's look at an example. Suppose you have a floppy disk named BASIC1 on which you've stored a number of BASIC programs. Your directory contains entries which are subdirectories pointing to other files. When you use the Filer program to get a directory of the disk, you might see something like this:

```
GAMES
HOME.FINANCE
ACCOUNTING
WORD.PROCESSING
COMMUNICATIONS
```

You could have stored all of your game programs under the subdirectory GAMES, all of your home finance files under the subdirectory HOME.FINANCE, all of your accounting files under ACCOUNTING, and so on, as shown in Figure 3-2. The more files you store on a disk, the more subdirectories become useful.

If you're storing large numbers of files on a hard disk, for example, you can organize specific files in layers of subdirectories. This whole thing is called a *hierarchical storage system*. We'll talk about how you create a subdirectory later in this chapter.

PATHS AND PATHNAMES

The Apple III filing system allows you to access a file directly, even if it's buried deep within several layers of subdirectories. You can go directly to a file by telling the computer which "path" to take. You do this by specifying a *pathname*.

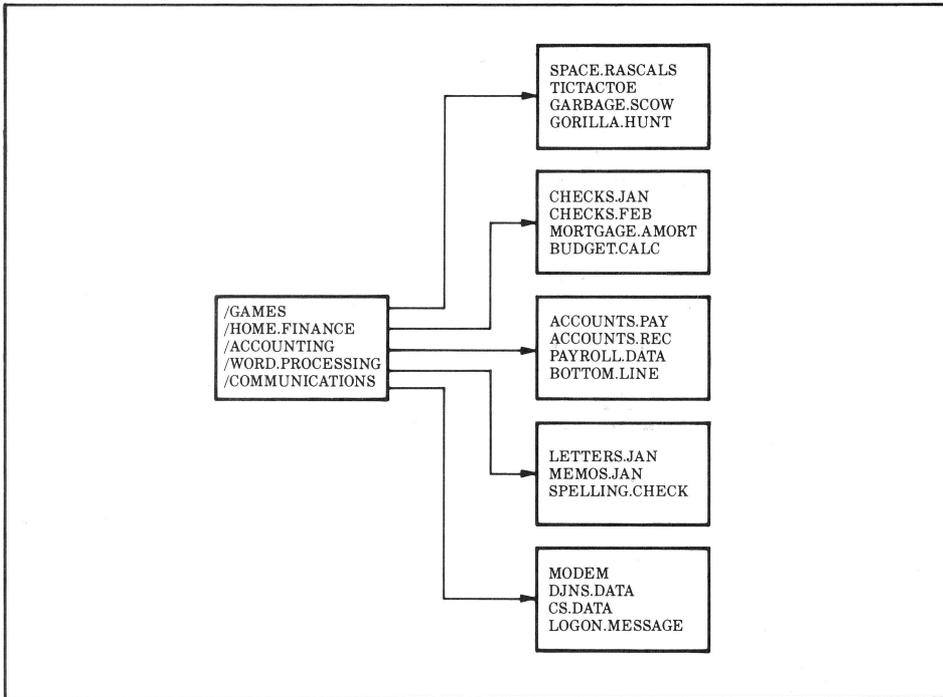


Figure 3-2. An example of subdirectories

A path is defined in one of two ways. Using the example shown in Figure 3-2, let's go directly to a specific game program (SPACE.RASCALS). This can be done by typing

```
/BASIC1/GAMES/SPACE.RASCALS
```

or

```
.D1/GAMES/SPACE.RASCALS
```

As you can see, each part of the path is preceded by a slash. The first part of the pathname is the *volume name*, the name of the disk drive where the file is located. Instead of the volume name, you can use the actual device name. Recall that device names always begin with a period. Apple III floppy disk drives are specified by .D1 through .D4. The built-in disk drive is always .D1. The Profile hard disk is referred to by the device name .PROFILE.

If you're going directly to a file that's on one of the drives and isn't filed under a subdirectory, your pathname can be relatively short. Here are a few examples:

```
.D1/TEXT.FILE.MAKER  
.D2/INSTRUCT.DOC  
.PROFILE/MORTGAGE
```

APPLE III UTILITIES

The dictionary defines the word "utility" as something designed or adapted for general use. That's exactly what your Apple III System Utilities disk is designed to do. It contains a number of programs that allow your computer to do "housekeeping" chores like displaying lists of files, formatting floppy disks, copying files, and so on. The System Utilities disk has quite a few programs on it, but essentially they're divided into three areas:

1. Operations on devices
2. Operations on files
3. Configuring the system.

Figure 3-3 shows the organization of the individual programs on the System Utilities disk.

The device handling utilities allow you to work with files on a device level; that is, they allow you to do file related things with entire devices. The device handling utilities are primarily used to do things like copy the contents of an entire disk drive.

File handling utilities, on the other hand, allow you to work with individual files on a disk. They allow you to do things like copy, delete, or rename individual files. The file handling utilities are referred to as the Filer.

The *System Configuration Program* is extremely important because it tells the Apple III's operating system what peripherals and accessories you have hooked up to your computer and defines how communications take place between the software and hardware. We will examine this in depth in Chapter 4.

Your Apple III comes with a floppy disk labelled System Utilities. In Chapter 2, you made a backup of that disk. Take the backup and boot it in the following manner:

1. Make sure the power is off.

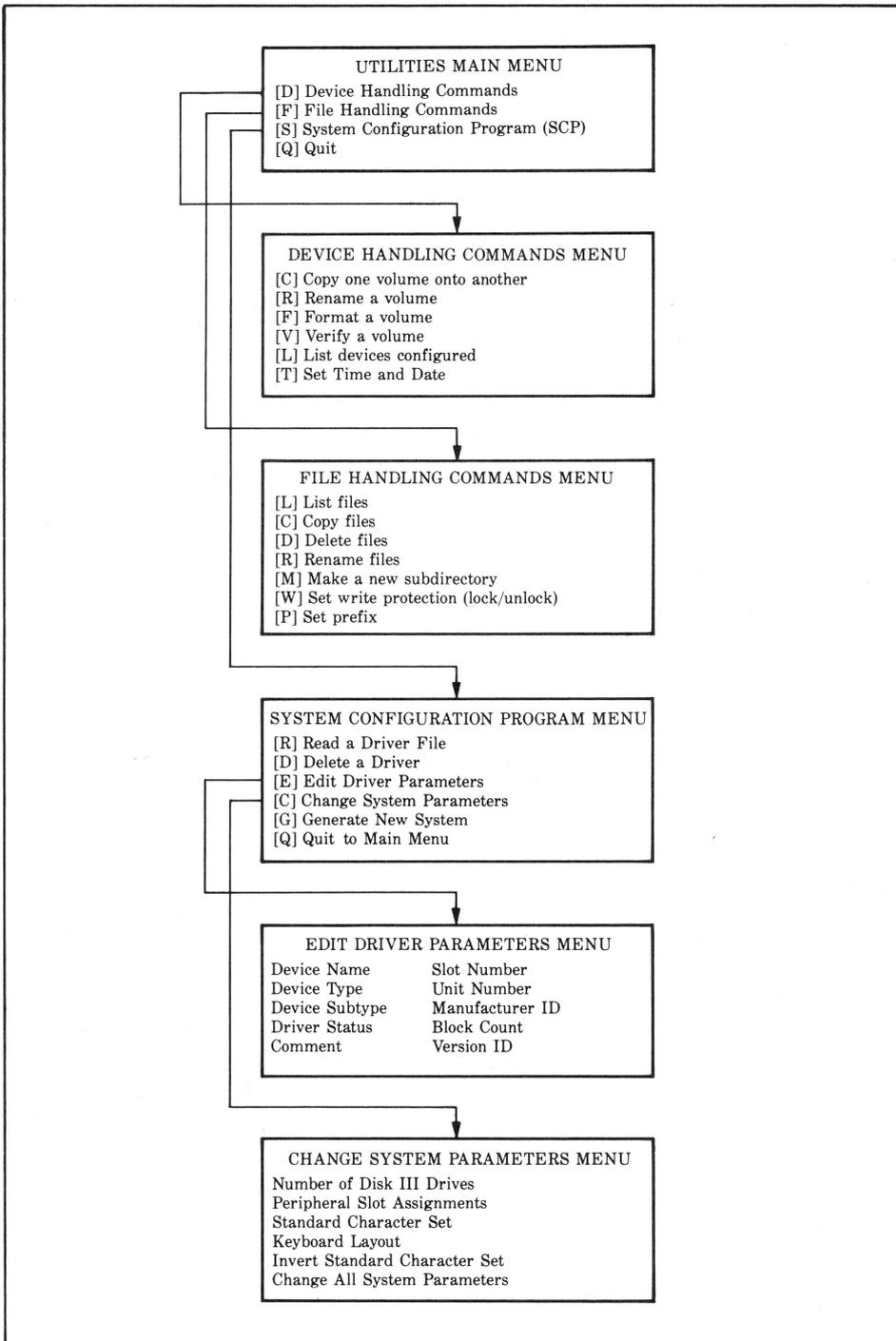


Figure 3-3. Programs on the Apple III System Utilities disk

2. Insert the disk into the built-in drive.
3. Turn the AC power on.

If everything works right, you should see the Utilities Main Menu (Figure 3-4) in about thirty seconds.

Working With Utilities

All the Apple III utilities that we'll talk about in this chapter work in essentially the same way. You'll see a menu (a list of the choices) and then you select an option from the menu in one of two ways:

1. Press the letter key corresponding to your choice.
2. Use the UP ARROW and DOWN ARROW keys to move the cursor to your choice. Then press RETURN.

In some cases, you'll see the message "Press RETURN to accept" on the screen. In this case, press RETURN to select the given option.

As you use the utilities, the Apple III will often ask you to enter information. A *prompt* will ask you for the information and a highlighted *field* will show you where you need to enter the information. In many cases, there will already be an entry in the field. This is the

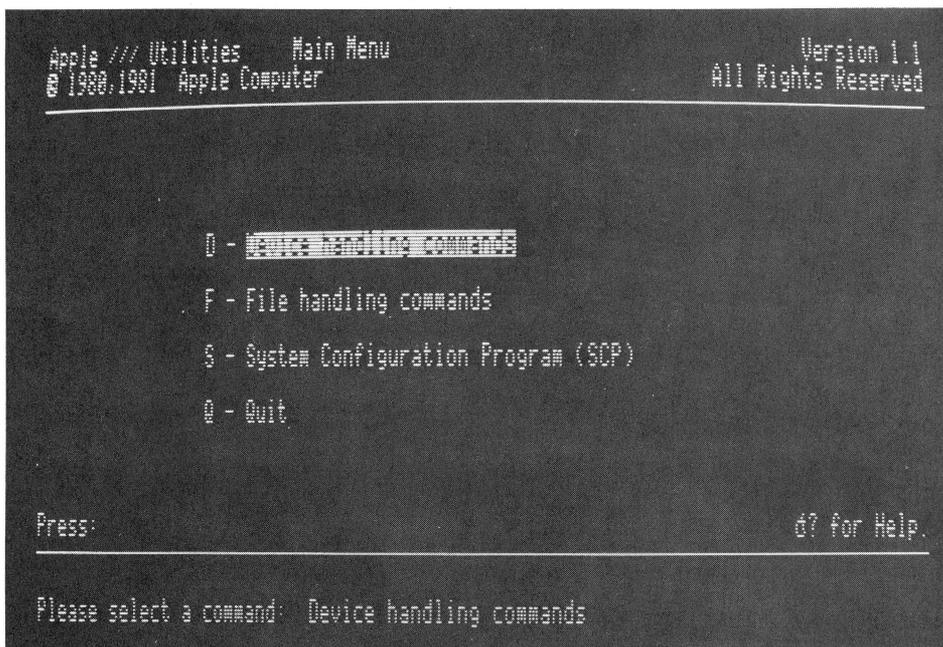


Figure 3-4. Utilities Main Menu

default, the response you'll be using most of the time. At this point, you have a number of choices:

1. If you want to return to the previous menu, press **ESCAPE**.
2. If you want to back up to a previous field, hold down the **CONTROL** key and press **RETURN**.
3. If the default is what you want to use, press **RETURN**.
4. If you need to change the default or enter additional information, type in the new information and press **RETURN**.

The utility disk has a built-in editor that lets you change or correct entries in the following ways:

1. You can use the **LEFT ARROW** and **RIGHT ARROW** keys to move the cursor to a mistake and type over it.
2. If you want to delete a character in the middle of an entry, use the arrow keys to move the cursor to the character you want to delete and press the **RIGHT ARROW** key while holding down the **OPEN APPLE** key. This will erase the character under the cursor and move the other characters to the left. If you press the **LEFT ARROW** key while holding down the **OPEN APPLE** key, the character to the left of the cursor will be deleted.
3. If you need to *add* characters to an entry, move the cursor to the position where you want to insert and press **I** while holding down the **OPEN APPLE** key. This puts you in the *insert mode*. Type the characters you need to insert and then *leave* the insert mode by holding down the **OPEN APPLE** key and pressing **I**.
4. If, while you're editing a field, you decide that you want to restore the default (if there was one), just press **ESCAPE** and the default will reappear.

Wildcards

Many of the Apple III utilities require that you enter pathnames. If you're typing a long list of files with similar names, you can avoid typing all the names by using a *wildcard*. In Apple III utilities, the equal sign (=) is the wildcard. It allows you to specify a group of filenames that are similar. For instance, you can specify a group of files that begin or end with a particular letter or all files that contain certain combinations of letters.

Let's use some examples. Suppose you wanted to specify *all* files that end in **.DATA**. You could do this by typing **/=.DATA**.

As another example, assume you have these files:

NOV83.SALES

DEC83.SALES

JAN84.SALES

FEB84.SALES

MAR84.SALES

You could specify all 83 sales by typing `=83.SALES`; or all 84 sales by typing `=84.SALES`. All of the files that end in `.SALES` could be specified with `=.SALES`.

THE FILER

The utility that allows you to work with Apple III files is called the *Filer*. To get to the Filer from the Utilities Main Menu, type `F` or move the cursor to the Filer entry on the menu and press `RETURN`. You'll see the File Handling Commands Menu, as shown in Figure 3-5.

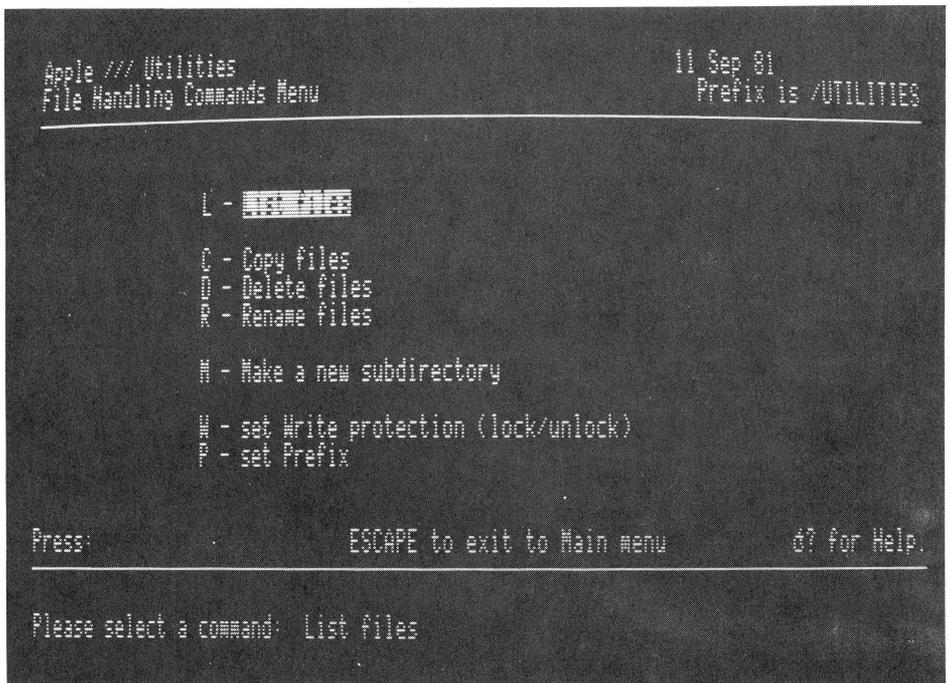


Figure 3-5. File Handling Commands Menu

Next, type `L`, or since the cursor is already positioned at the “List files” entry, just press `RETURN`. The Apple III will display this message at the bottom of the screen:

```
List the directory information of the files:
.D2
Including All directory levels; sending the listing to the file:
```

The *default* name of the disk to list is `.D2` (the first add-on drive). If that’s the one you want to list, just press `RETURN`. Otherwise, if you want a list of files that are on the built-in disk, type `.D1` and press `RETURN`.

The Filer next asks you for the directory levels you want to list. If you want all levels of files, press `RETURN` again. You can also choose to see just the first level of files (subdirectory names) by typing a `1` in this space.

The final option asks you where you want the directory listing sent. The default is `.CONSOLE`—the video display. You can also send the directory to a disk file by typing a pathname such as

```
.D1/DIRECTORY.LIST
```

Alternately, you might want to send your directory listing to a printer. Depending on which device driver you have installed for printing, you’ll likely type one of the following:

```
.PRINTER
.PARALLEL
.QUME
```

An example of a directory listing to the screen is shown in Figure 3-6.

The *volume name* of the disk appears at the top right along with the date and time. Below the long horizontal line is the actual list of files on the disk. As you can see, each directory entry consists of the filename, the file’s size in 512-byte blocks, the date and time it was last changed, the file type, and several numbers pertaining to how the file is stored on the disk.

Finding Files Quickly

Most of the options in the Filer require you to enter a filename or pathname.

If you are not sure of the filename, you can enter the disk name and

press the UP ARROW or DOWN ARROW key and a box will appear that contains a list of the files on the disk you've selected. The cursor will move up into the box and you can use the UP ARROW and DOWN ARROW keys to point to the file or files you want. To mark a file, press the RIGHT ARROW key and an arrow will appear next to the filename. (If you change your mind, pressing the LEFT ARROW key will remove the marker.) To mark additional files, move the cursor up or down and press the RIGHT ARROW key again. Once you've marked all the files you need, press RETURN and all the files you've marked will be selected for use by the utility you are using.

Copying Files

As you saw when you first entered the File Handling Commands Menu, you can do quite a few more things than just list the files that you've stored on a disk. To return to the Utilities Main Menu, press ESCAPE. Let's look at the other options.

The second option on the filer menu is "Copy files." Since the filer utility works with *individual* files, you'll use the "Copy files" option to copy individual files. If you want to copy *everything* on a disk, you use a device handling utility.

To get to the Copy files utility from the File Handling Commands Menu, type c or move the cursor to the "Copy files" entry and press RETURN. At the bottom of the screen, you'll see a message like this:

```
Copy the files:
[
To the files
```

If you only have a single disk drive (built-in), you should insert the disk that you want to copy from into the drive. (If you have a second drive, place the source disk there.) Should you be copying a file from a Profile hard disk, you won't have to insert anything.

If you're not sure of the exact name of the file you want to copy, enter the name of the disk and then press the UP ARROW or DOWN ARROW key. You can then choose the file you want to copy as described in the last section.

If you only want to copy a single file, all you have to do is enter its pathname and press RETURN. Then enter the pathname of the file you want to copy to (if it's different than the original). When you press RETURN, the copying process will begin.

If you use the arrow keys to select a number of files, you'll see that an equal sign (=) appears next to the message "Copy the files." This is a

wildcard that tells the system that you want to copy *all* the files you've marked.

If you have a second disk drive, you'll also notice that the Apple III enters .D2 as the *default* destination device. (It assumes that you want to copy to the second drive.) If you only have a single drive, you'll see .D1 as the default drive.

When everything is set up the way you want it, press RETURN and the copy will proceed. As each file is copied a message will appear in the middle of the screen.

Existing Filenames

If in the process of copying files, the copy utility finds that the destination disk contains a file with the same name as the file you are copying to, you'll hear a beep and get the message:

```
Delete old/volume name/filename?(Yes/No)
```

Copying to a file with the same name will *erase* the old file, so make sure you want the old file erased before you press Y. If you press N you can start the copy operation once again.

Copying With a Single Disk Drive

Finally, if you are using the same disk drive for both the source and destination disks, you'll be required to change disks a number of times. When it's time to change disks, you'll see the messages

```
Insert Source volume
```

```
Type SPACE to continue; ESCAPE to quit
```

or

```
Insert Destination volume
```

```
Type SPACE to continue; ESCAPE to quit
```

Deleting Files

The "Delete files" option of the Filer erases individual files or groups of files. Choosing a file or files to erase is exactly the same as in the copy option. Once you're prompted for the name of the files to delete with the message

Delete the files:

[

you can either type in individual file names or enter the name of the disk drive and press the UP ARROW or DOWN ARROW key to choose from a list of files.

Once you've selected the files to delete, you'll see the following message:

```
File filename      deleted
Update directory ? [Yes/No]
```

Pay close attention. Even though the message *says* the file is deleted, it's really still there and will remain so until the disk's directory is updated. If you've changed your mind about erasing the file, type N. Otherwise, type Y and the file will actually be deleted.

NOTE: After you've erased something, there's nothing you can do to get it back. So be sure before you type Y.

Deleting Protected Files

Files that are *write-protected* can't be erased until they're "un-protected." If you try to erase a protected file, you'll see the message

```
filename -- Write protect error
Type SPACE to continue; ESCAPE to quit
```

When you see it, you'll have to press ESCAPE, which returns you to the main Filer menu. (See the section on "Write-Protection" later in this chapter.)

Renaming Files

The "Rename files" option of the utilities disk lets you change the name of files you've created. Do *not* change the names of system files which are supplied by Apple. The Apple III expects to find certain files. If you rename the system files, you'll end up with a system that doesn't work correctly. In particular, don't rename the following files on your System Utilities disk:

SOS.KERNEL

SOS.DRIVER

SOS.INTERP

SYSTEM.PASCAL
 SYSTEM.MISCINFO
 SYSTEM.STARTUP

Although you can rename files using the wildcard (=) character, we suggest that you rename files individually.

Creating Subdirectories

As we discussed earlier in this chapter, a subdirectory is a file containing the names and addresses of additional files. Subdirectories are appropriate for organizing files into logical groups.

When you choose the "Make a new subdirectory" option, you'll be asked for the name of the subdirectory with the following message:

```
Make a subdirectory called
[/UTILITIES
with enough space for    25    files.
```

To review, remember that a pathname starts with the volume name of the disk. The Filer fills in the name of the disk in the built-in drive (in this case /UTILITIES). Suppose you want to create a subdirectory named "GROUP1" on the utilities disk. Move the cursor to the end of /UTILITIES, type /GROUP1, and then press RETURN twice.

You can also create a subdirectory by using a device name. Here are some examples of subdirectories on various devices:

```
.PROFILE/GROUP1
.D2/GROUP1
/BASIC/GROUP1
```

The default value for the number of files in a subdirectory is 25. That's enough for most subdirectories; however, if you think you'll need more or less, be sure to change the number. Don't create a large subdirectory if you don't need to, since the space for a subdirectory takes up space on your disk. However, if you create a subdirectory that has too few files, you won't be able to expand it.

If you try to create a subdirectory that already exists, you'll see a message asking you if you want to delete the old subdirectory. Be sure this is what you want to do before typing Y, since all files in the old subdirectory will be erased.

Write-Protection

Although you can protect the entire contents of a disk from being erased or changed by putting a write-protect tab over the notch on the disk, you can also protect individual files by “locking” them with the “Set Write protection (lock/unlock)” option of the Filer. When you lock a file, you can read the data in that file, but you can’t erase it or change it.

You can lock or unlock any or all files on a disk. In fact, it’s a good idea to make sure the six system files we listed in the last section are locked so that you won’t *accidentally* erase or change them.

Once again, you can either type the names of individual files to lock or unlock or you can enter the name of the disk and press the UP ARROW or DOWN ARROW keys to select from a list of files. The message

```
Turn Write protection on? [Yes]
(Yes to lock, No to unlock)
```

will appear in the middle of the screen. Typing **Y** will lock all the files you’ve specified. As they’re locked, you’ll see the files listed on the screen.

The process of unlocking files is nearly identical to locking them except you press **N** to unlock the files you’ve selected.

Finally, it’s important to note that if you try to lock a file that’s already locked or unlock files that are already unlocked, there won’t be any change in their status.

Set Prefix

The *prefix* is a special pathname that is stored in the Apple III. It allows you to enter the local names of files without having to specify the entire pathname. When you boot up from disk, the prefix is set to the volume name of the disk. You can use the “Set prefix” option of the Filer to change the prefix.

Whenever you enter a filename that does not begin with a “/” or “.”, the filename is automatically appended to the prefix. For example, if you are using a disk whose volume name is /KEYS and you wish to specify a file described by the pathname /KEYS/HOUSE/NUMBERS, you would only need to type **HOUSE/NUMBERS** since the volume name /KEYS was assigned to the prefix when you booted from the disk. If you use the “Set prefix” option to change the prefix to **KEYS/HOUSE**, you would only need to type **NUMBERS** to specify the file.

Once you have set a prefix, you can override it by simply entering a complete pathname, since the prefix is not used when you enter a name beginning with a “/” or “.”.

DEVICE HANDLING UTILITIES _____

The first option in the Utilities Main Menu is “Device handling commands.” Whereas the file handling commands we talked about in the last section work on individual files, the device handling commands work on entire devices—in particular floppy and hard disk drives.

Copying Volumes

We talked about this option extensively in Chapter 2 when we explained how to make a backup disk. This option allows you to copy the *entire* contents of one disk to another. (If you need a review, see Chapter 2.)

Renaming Volumes

The “Rename a volume” option allows you to change the volume name of a disk. When you select this option, you’ll see the message

```
Rename the volume:
.D2
With the new volume name:
[           ]
```

Once again, the utility supplies a default device name (.D2) which you can change if you wish. If you want to change the volume name of the disk, you can type in the new name (beginning with a slash) or modify the existing name.

Formatting Disks

The format option, which was also explained in Chapter 2, prepares a blank disk for storing information by dividing it into *tracks* and *sectors*. In most cases, you need to format a disk before you can use it.

The format option asks you for the volume name of the disk to format. Once again, the volume name must conform to the conventions for Apple III volume names. If you don’t enter a volume name, the format utility automatically names the disk “BLANK” followed by a number. Later you can change it using the rename utility.

Verifying Volumes

The “Verify a volume” option lets you check a disk when you’re not sure everything is right. For instance, if you start getting errors in a program you’ve been running for a long time certain areas on the disk may have worn out or have otherwise been destroyed. The verify utility checks the disk without harming any of the data that’s already stored on it.

When you use this option, the Apple III will check the disk you specify. If there are no bad areas (blocks) on the disk, you’ll see the message

```
0 bad block(s)
```

at the top of the screen. Otherwise, the numbers of the bad blocks will be listed. If you do find bad blocks, copy all the files from the defective disk to a new disk. Remember though that the files stored in bad areas still won’t work correctly on the new disk. That is why you should keep a backup of original files.

To avoid problems, it’s a good idea to verify disks after you’ve formatted them and *before* you use them to store data. In addition, don’t over-use disks. If you have files that you use nearly everyday, put them on several disks and “rotate” them. Disks do wear out, and that’s why it’s essential to buy the best you can afford.

Listing Configured Devices

The “List devices configured” option gives you a list of the *device drivers* that may be used with a particular disk. As we’ll see in detail in the next chapter, device drivers are programs that allow your programs to access a device. The list of devices that you may access from a particular disk depends on what device drivers have been installed on the disk. Once you choose the “List devices configured” option, you can specify where you want the list sent. `.CONSOLE` is the default device. If you just press `RETURN`, you will see the list of devices on the screen. We’ll explain this in more detail in Chapter 4.

Setting Time and Date

The final option in the Device Handling Commands Menu is “Set Time and Date.” If you want to record the time and date when you create a file with the System Utilities disk, you need to first set the time and date with this option.

It's important to remember that since the Apple III doesn't have an internal calendar, it doesn't keep track of the time and date. Therefore, if you want the correct date and time, you'll need to set it *each time* you want to use it.

System Configuration Program

4

One of the most powerful features of your Apple III is its ability to utilize a wide variety of accessories and peripherals. This power comes from the Sophisticated Operating System (SOS). This chapter will give you an introduction to SOS and show you how to customize SOS for use with the peripherals you purchased with your system.

SOPHISTICATED OPERATING SYSTEM _____

There are two types of software: *applications* software and *system* software. Applications software allows your Apple III to do specific tasks. All the programs on your System Utilities disk are applications, as are programs like word processors, accounting packages, and even games. But an application program can't entirely control the Apple III's microprocessor, memory, associated circuitry, and peripherals. For that you need system software—otherwise known as an *operating system*. You can think of the Apple III's Sophisticated Operating System as the “traffic cop” of the Apple III, coordinating the flow of data between the microprocessor, memory, and peripherals like disk drives. One of its most important jobs is servicing the keyboard whenever you press a key and putting messages on the video display to let you know what is happening.

Although a full technical explanation of SOS is well beyond the scope of this book, an overview will be helpful in understanding how the Apple III works.

Parts of SOS

There are five basic components to SOS:

- The *file manager* controls the storage and movement of files (collections of data).
- The *device manager* controls the storage and flow of data to and from the keyboard, video screen, printer, disk drives, and other devices.
- The *memory manager* allows the Apple III's 8-bit microprocessor, which is normally limited to addressing 64K of RAM, to address up to 256K bytes.
- The *interrupt manager* allows devices like the keyboard and serial input/output port to temporarily suspend normal processing when they require servicing. For example, when a character is ready for input from the keyboard, the interrupt manager can interrupt the current program to allow the character to be read.
- The *utility manager* controls the system clock, schedules events, and lets you use the joystick interfaces.

SOS is loaded into memory every time you boot the system. A boot disk *must* contain the two files that comprise the operating system: SOS.KERNEL and SOS.DRIVER. SOS.KERNEL contains the core of SOS—the parts that must always be in memory regardless of what type of system you have. SOS.DRIVER is the file that we'll be detailing in this chapter. It contains specific programs that you customize for the peripherals you have in your system.

Using SOS

SOS is unique among personal computer operating systems because you can't communicate with it *directly* using commands. Instead, you must use applications software to communicate *indirectly* with SOS. The most common examples are the utility programs on the System Utilities disk.

DEVICE DRIVERS

SOS uses *device drivers* to communicate with the various parts of your system. Device drivers are special programs that act as “go-betweens” for the hardware (keyboard, screen, printer, disk drives, modem, and so forth) and the operating system. In order to use any of

these peripherals, you must have the correct device driver installed on the *boot disk*—the disk that you use to boot from.

Although it's useful to think of device drivers as separate programs, to be absolutely correct you should note that device drivers are actually *part* of your Apple III's Sophisticated Operating System. (See Figure 4-1 for a visual representation.)

The following is a list of the most common device drivers. Note that the four drivers for the floppy disk drives are listed separately but are actually contained in one module.

.FMTD1	Built-in disk drive
+FMTD2	Additional disk drives (the plus sign indicates that the drivers are tied together)
+FMTD3	
+FMTD4	
.CONSOLE	
.PRINTER	RS-232-C port (out only)
.SILENTYPE	Apple Silentype printer
.PROFILE	Profile hard disk

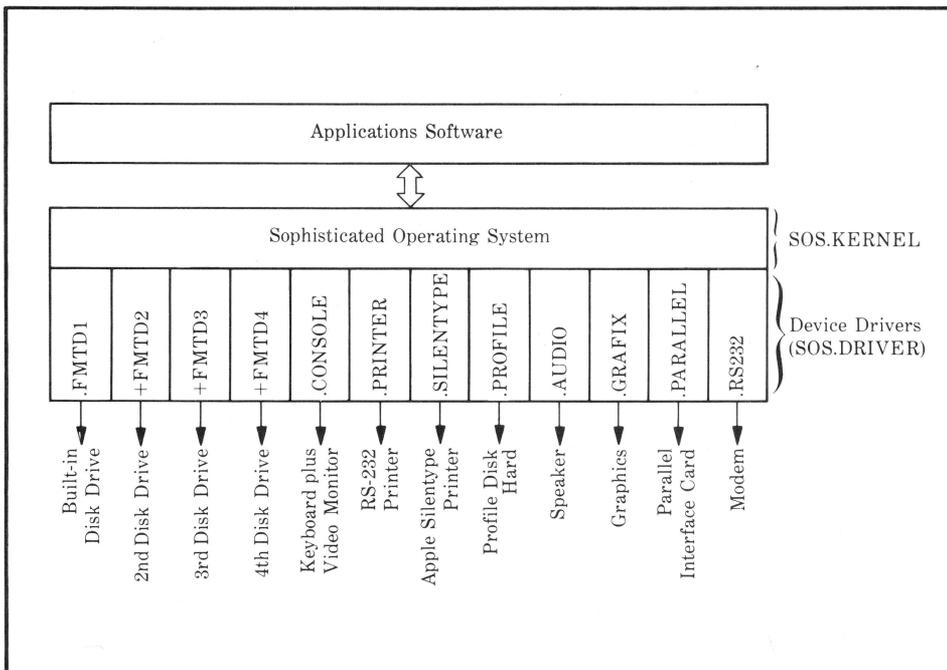


Figure 4-1. SOS device drivers

.AUDIO	Built-in speaker
.GRAFIX	Apple III graphics
.PARALLEL	Parallel interface card
.RS232	RS-232-C (2-way for a modem)

If your dealer installed the device drivers before you received your Apple III, you won't need to do any modifications in order to use your system. However, if you buy new peripherals, new software, or sell some of the parts of your system, you'll need to know how to install the right device drivers on your disks. That's where the System Configuration Program (SCP) comes in. With it, you can add, delete, or modify individual device drivers to "customize" the operating system for the hardware that you have.

Size Limitations

You might wonder why all Apple III disks don't come with all of the possible device drivers installed. Device drivers take up quite a bit of space on a disk. In some cases you'll even need to have two disks to get your Apple III up and running. The bottom line is that you should have the fewest possible number of drivers on a disk.

SYSTEM CONFIGURATION PROGRAM _____

You use the System Configuration Program (SCP) to customize your system and add, delete, or change device drivers. The SCP is contained on the disk marked System Utilities. (Make sure you make a backup of the original System Utilities disk and put the "master" away in a safe place before you go any further.)

What you want to do is make a customized system utilities disk that contains all the device drivers for your particular Apple III. Once that's done, you can copy the SOS.DRIVER file to any of the other boot disks you'll be using with your system.

Finding the SCP

To get started, put the System Utilities disk into the Apple III's built-in disk drive and turn the system on. It takes about thirty seconds for the Apple III to load all the files it needs into its memory. When it's done, you'll see the Utilities Main Menu, shown in Figure 4-2.

```

00000000 // Utilities Main Menu
00000000 Apple Computer
00000000 Version 1.1
00000000 All Rights Reserved

D - Device handling commands
F - File handling commands
S - System Configuration Program (SCP)
Q - Quit

Enter Pathname of Driver File:

```

Figure 4-2. Utilities Main Menu

We explained device and file handling commands in Chapter 3. To get to the SCP, either type `s` or use the UP ARROW or DOWN ARROW keys to move the cursor to the SCP entry and press RETURN. The SCP menu is shown in Figure 4-3.

Reading a Driver File

Before you can modify a device driver, you'll have to read the contents of `SOS.DRIVER` from a disk. (If you try to use any of the options other than "Quit to Main Menu", you'll hear a "beep" from the Apple and receive a message saying that no device drivers have been read.)

To read a driver, type `R` or move the cursor to the entry "Read a Driver File" and press RETURN. This will result in the screen shown in Figure 4-4.

At the bottom of the screen, under the message "Enter Pathname of Driver File:", you'll notice a default entry—`.D1/SOS.DRIVER`. This tells the SCP to read the device driver file from the built-in disk drive (`.D1`). If you want to read the driver from a different drive, you can

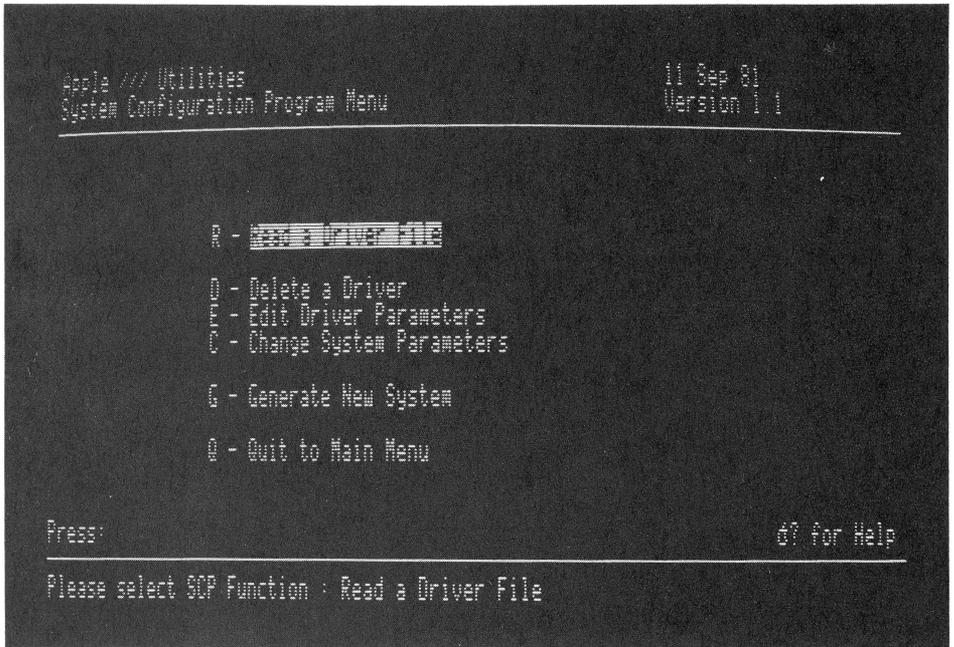


Figure 4-3. System Configuration Program menu

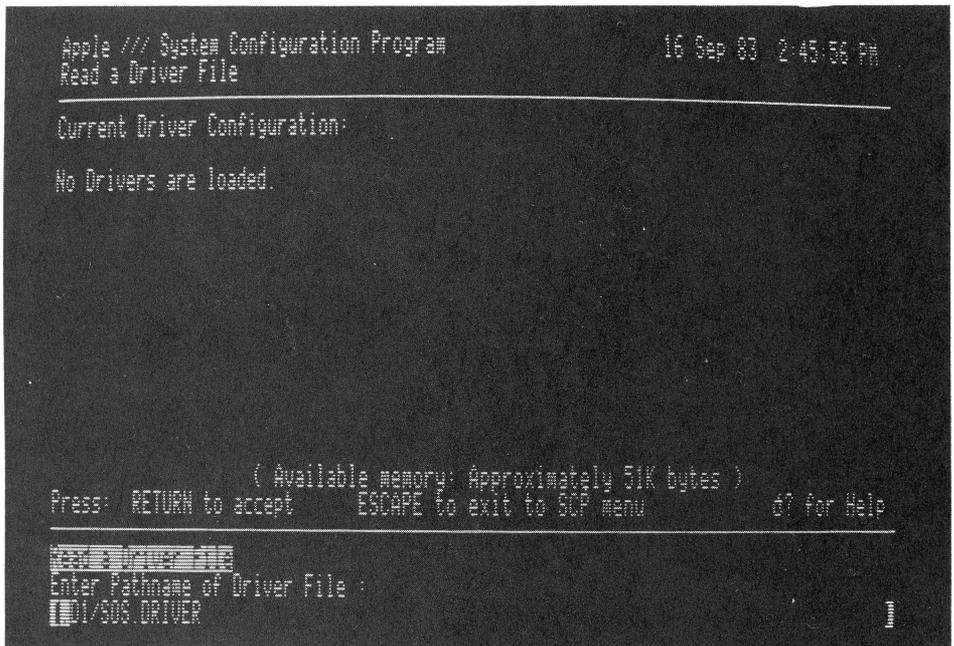


Figure 4-4. Reading a driver file

```

// System Configuration Program
// Driver File
16 Sep 83 2:45:36 PM

Current Driver Configuration:

(available memory: Approximately 32K bytes)
Press: RETURN to accept ESCAPE to exit to SCP menu 0? for help

D2/SOS.DRIVER

```

Figure 4-5. List of device drivers

change the default entry. For example, you could enter **.D2/SOS.DRIVER**.

NOTE: The file that contains all the device drivers is *always* named **SOS.DRIVER**.

To read the device drivers from the disk in the built-in drive, just press **RETURN**. The Apple III needs a few seconds to read the driver file, and then you'll see a screen similar to the one shown in Figure 4-5. (The exact list of device drivers will depend on what drivers were installed by your dealer.)

Note that the four floppy disk drivers are linked together into one *module*. Even if you have only one disk drive (the built-in unit), you'll still have all four device drivers in your **SOS.DRIVER** file.

WORKING WITH DRIVERS

Once you have loaded a device driver file into the Apple III, there are a number of things that you can do with it:

- Load additional drivers
- Delete drivers

- Edit drivers
- Leave things the way they are.

If your device driver file was already tailored for your system by your dealer, you won't need to do anything else. If you don't want to change anything, press ESCAPE and then Q to return to the Utilities Main Menu. Once you do this, you'll see a message that says

GENERATE not performed. Quit anyway? [Yes/No]

If you're sure you want to exit the SCP after going to the trouble of loading a device driver, press Y, and you'll be returned to the Utilities Main Menu; if you press N, you'll be returned to the screen that contains the list of device drivers.

Loading Additional Drivers

Most of the time, you'll be using the System Configuration Program when you want to add or delete device drivers to or from the SOS.DRIVER file. For instance, if you purchase a parallel interface card, a modem, or a Profile hard disk, you'll need to add the appropriate drivers to the SOS.DRIVER file in order to use them.

Let's use an example. Suppose you've purchased a Profile hard disk and wish to add its driver to your SOS.DRIVER file. You'll be supplied with a disk marked "Profile Driver and System Utilities Software." You can find the name of the file that contains the Profile device driver by using the "File handling commands" option on your System Utilities disk and pressing L (for List files). The name of the driver file always ends with .DRIVER. In this case, the name of the driver file is PROFILE.DRIVER.

To add the Profile device driver to the drivers we listed previously, return to the SCP menu and select the "Read a Driver File" option once again. You'll see the list of device drivers that we read from the System Utilities disk. Next, if you only have a single disk drive, you'll need to remove the System Utilities disk from the built-in drive and replace it with the Profile Driver and System Utilities Software disk. If you have an additional drive, you can place the Profile Driver disk in the second driver. In response to the prompt "Enter Pathname of Driver File:" type either

.D1/PROFILE.DRIVER (Single-drive system)

or

.D2/PROFILE.DRIVER (Two-drive system)

You'll see the driver named `.PROFILE` added to the list of device drivers. But even though it's added to the list, we still haven't created an updated `SOS.DRIVER` file. We'll explain that in the section called "Generating the System" later in this chapter.

Deleting Drivers

As we mentioned earlier, there's no sense keeping drivers that you'll never use in the `SOS.DRIVER` file. For instance, if you don't have a Silentype printer, you can delete `.SILENTYPE`; or if you don't have a serial printer, you should delete `.PRINTER`.

Deleting a driver is a simple matter of selecting the "Delete a Driver" option from the SCP menu. Type the number of the driver you want to delete and press `RETURN`. The program will ask you if you're sure you want to delete the driver. If you press `y`, the driver will be deleted and you'll see an updated list of drivers. (Remember that anytime you change *anything* in the `SOS.DRIVER` file, you'll need to generate a new system. This will be explained in detail later in this chapter.)

Editing Driver Parameters

Individual device drivers contain *parameters* that specify how a driver will interact with a device. In most cases, you won't need to change the parameters. In fact, we suggest that you don't change the parameters of individual drivers unless you're *absolutely* sure you know what you're doing. This task is best left to your dealer.

To see what the parameters look like, select the "Edit Driver Parameters" option from the SCP menu. Type the number of the individual driver you want to edit and press `RETURN`. Let's use `.PROFILE` for our example. When you select the `.PROFILE` driver for editing, you will see the screen shown in Figure 4-6. Let's look at what the individual parameters mean.

Device names. The device name is the name of the device that the driver communicates with. Although you probably won't ever need to change the name, note that the name in the parameter list must conform to the rules for a SOS filename. To review, a device name must begin with a period, can be up to 15 characters long, and can contain only letters, numbers, and periods. If you try to type in an illegal name, SOS will give you an error message.

NOTE: Although you can use any device name you wish, many Apple

```

APPLE III /// System Configuration Program                               16 Sep 81 2:45:58 PM
Driver Parameters

-----
Item #   Field                               Value
-----
1 - Profile Name ..... PROFILE
2 - Device Type ..... 00          Block, Read, Write (range: 00..FF)
                                     Formatter present; NonRemovable
3 - Device Subtype ..... 02          (range: 00..FF)
4 - Driver Status ..... ACTIVE
5 - Comment
   APPLE /// SOS Profile Driver (C) Copyright 1981, Apple Computer Inc.
6 - Configuration Block data

Slot Number ..... 04
Unit Number ..... 00
Manufacturer ID ..... 0001 Apple
Block Count ..... 2688 (5720)
Version ID ..... 00

Press: F10 to exit to top Edit menu  or  for Help
-----
Driver Parameters
Select an item to be edited: Device Name

```

Figure 4-6. .PROFILE driver parameters

III programs use the default name. If you rename a device, you could encounter problems later on.

Device type and subtype. These parameters specify how SOS will treat the device. Both device type and subtype are hexadecimal (base 16) numbers. In most cases, you will not need to change this information. The only time you'll need to know about this information is if you're writing your own device drivers, something beyond the scope of this book.

Driver status. Any device driver can be *active* or *inactive*. If it's inactive, SOS does not load it into memory when you boot the system. Although inactive, a driver still takes up the same amount of memory space on your floppy disk. In most cases, it's better to eliminate a driver from your working disk than to make it inactive. However, if you have drivers that you want to use occasionally, you can make them inactive. To do this, select the "Driver Status" entry (item 4) and press RETURN. You'll see the screen shown in Figure 4-7.

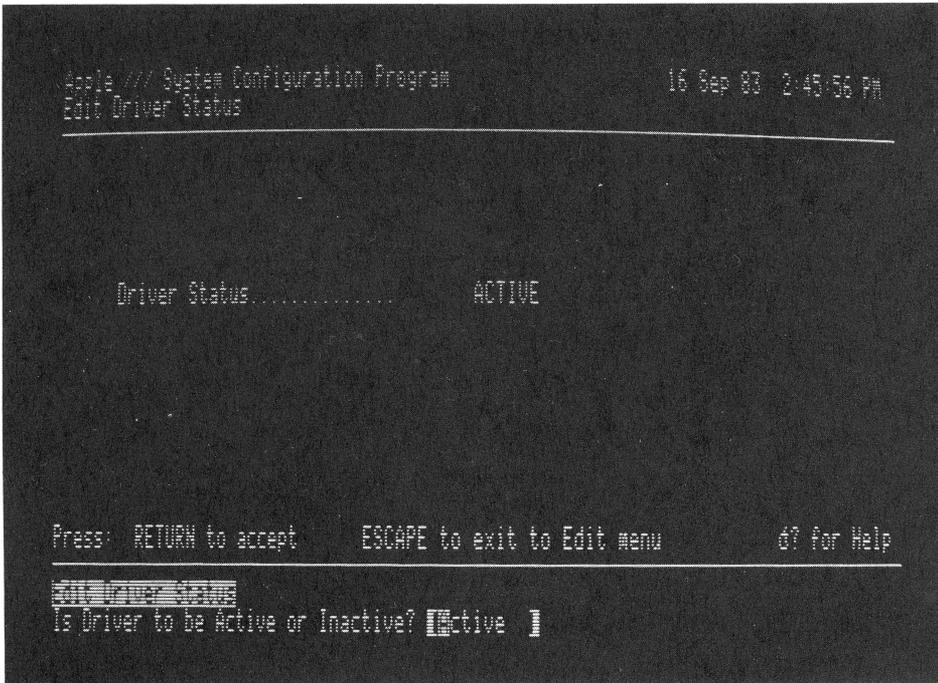


Figure 4-7. Editing driver status

Type **INACTIVE** in response to the prompt and press **RETURN** to make the driver inactive. To make an inactive driver active, follow the same steps and enter **ACTIVE** in response to the prompt.

Comments. The *comment field* is simply an area where you can enter as many as 80 characters to remind you what the driver is and how to use it. Drivers supplied by Apple, such as **.PROFILE**, normally contain copyright information. Usually you don't need to change this information, although you can insert the date that your driver was installed.

Configuration block. The last item in the list of parameters is the configuration block. Note that not all devices have configuration blocks. The configuration block allows you to customize the driver to the device you're using. Although you'll probably seldom need to change this information, you should be aware of what it looks like. Select the "Configuration Block Data" entry (item 6) and you'll see what the configuration block looks like. Figure 4-8 shows the configuration block for the **.PROFILE** driver.

Miscellaneous information. Below the list of driver parameters is miscellaneous information that is set by the SOS software. Although it is possible to change these values, changing anything other than the slot number requires advanced programming skills.

Slot number. The number of the expansion slot that the interface circuit board plugs into is called the slot number. (Remember that the slots are numbered 1 through 4, from left to right as you look from the front of the Apple III.) If the device driver doesn't use an expansion slot (as is the case with floppy disk drives), the slot number is marked "n/a."

NOTE: You can change the slot numbers of interface cards by using the "Change System Parameters" option of the SCP (see the next section).

Unit number. Each module of a modular driver, such as the one for the four floppy disk drives, is given a unit number starting with \$00. Drivers that aren't modular all have the \$00 unit number.

Manufacturer ID. This is a hexadecimal number that identifies the company that wrote the device driver. All Apple software has the number \$0001. Apple assigns other numbers to other manufacturers.

Version ID. This is the version number of the driver software. By convention, version numbers are three digits long. Depending on when you purchased your Apple III, you'll see 1.10, 1.20, 1.30, or a higher number.

APPLE III SYSTEM PARAMETERS ---

The device parameters control how the operating system interacts with a device. As you might guess, the system parameters control the operation of the Apple III system as a whole. You can set four system parameters:

- The number of floppy disk drives you have
- The style of characters that are displayed on the screen
- The way the keys are arranged on the keyboard
- The location of the interface cards in the expansion slots.

To take a look at the system parameters, select the "Change System Parameters" option from the SCP menu. You'll see a screen that looks something like the one shown in Figure 4-9.

You can change any of the system parameters by selecting the parameter you want to change and following the prompts. Here's what each of the system parameters means:

```

Apple /// System Configuration Program          19 Sep 83  2:45:35 AM
Change System Parameters

-----
CURRENT SYSTEM PARAMETERS:

- Number of Disk III Drives ..... 2
- Peripheral Slot Assignments
- Standard Character Set ..... STANDARD
- Keyboard Layout ..... SHOLES

- Invert Standard Character Set
- Change All System Parameters

Press:          ESCAPE to exit to SCP menu          d? for Help

-----
Change System Parameters
Select Parameter to be Changed: Number of Disk III Drives

```

Figure 4-9. System parameters

Number of Disk III drives. This number should match the number of floppy disk drives you're using. Remember that the built-in drive counts as one.

Peripheral slot assignments. This parameter lets you assign slot numbers for expansion circuit boards. In most cases, you won't need to change the existing slot numbers; however, if you ever need to install a circuit board in a slot that is already occupied, you will need to reassign slot numbers.

To assign a new slot number, move the cursor to the "Peripheral Slot Assignments" entry and press RETURN or simply type 2. You'll see a list of the device drivers you've loaded and the slots that are assigned to the corresponding expansion boards. (Device drivers that don't require expansion boards are marked "n/a.") You can change a slot assignment by selecting a driver and filling in the new slot number in response to the prompt.

Standard character set. The "standard" character set is what is normally displayed on the screen. There are, however, three other character sets: Apple, Roman, and Byte. You'll find the files that allow you to use these typestyles on the disk marked "System Utilities Data."

Which typestyle you use is strictly a matter of personal preference. If you'd like to experiment with the other available styles, your Business

BASIC disk contains a BASIC program called FONTDEMO that demonstrates the different styles. You can change typestyles by choosing the "Standard Character Set" option of the system parameters menu and then typing the pathname of the file containing the new character set. For example, to change the typestyle, select the "Standard Character Set" option from the menu of system parameters. Then place the System Utilities Data disk in the add-on drive and type

```
.D2/FONTS/APPLE
```

or

```
.D2/FONTS/ROMAN
```

or

```
.D2/FONTS/BYTE
```

or

```
.D2/FONTS/STANDARD (To return to "normal")
```

If you only have a single drive, place the System Utilities Data disk in the built-in drive and type .D1 and the name of the file. Remember that your Apple III won't start using the new typestyle until you boot from a disk with the new SOS.DRIVER installed.

Keyboard layout. The actual assignment of keys on the Apple III keyboard is defined by SOS. The standard typewriter keyboard, sometimes known as the QWERTY keyboard, is referred to as "SHOLES" in the menu of system parameters.

You can change the layout using the menu of system parameters. Your System Utilities Data disk contains a file that lets you redefine the keyboard according to the Dvorak American Simplified Keyboard layout. You can load this file by placing the System Utilities Data disk in the second disk drive, selecting the "Keyboard Layout" option, and typing

```
.D2/KEYBOARD.LAYOUT/DVORAK
```

If you have only one drive, place the System Utilities Data disk in the built-in drive, select the "Keyboard Layout" option, and type

```
.D1/KEYBOARD.LAYOUT/DVORAK
```

When you boot from a disk with this driver installed, your keyboard will produce characters according to the Dvorak layout. (You'll still be

left with the problem of changing the keycaps to correspond to the new layout.)

You can also set up the keyboard to produce characters needed by other languages like French or Spanish. The software needed for these changes isn't included with your Apple III. You can see your dealer if you're interested.

Invert standard character set. This option lets you alter the character set that you loaded with the "Standard Character Set" option so that characters are displayed as dark characters on a light background.

Change all system parameters. This final option lets you read all of the system parameters from another disk containing an SOS.DRIVER file. Once you select this option, type in the pathname of the file containing the system parameters you want to use.

GENERATING THE SYSTEM

Once you've made all the additions, deletions, and changes to the device drivers, you can *generate* the new system. When you select the "Generate New System" option from the SCP menu, the Apple III will attempt to verify that the changes you have made to the driver file are valid. Once the verification is complete, you can enter the pathname for the driver file you want to generate. If there is no other file by that name on the disk that you specified, the drivers and parameters that you have set up are loaded into the file.

If there's already a file with the same name, you'll be asked if you want to erase the old file and create a new one.

Once the system has been generated, the Apple III will return to the Utilities Main Menu. Remember that in order to use the newly generated system, the drivers must be stored under the name SOS.DRIVER on the disk that you boot from. Make sure that all the disks you'll be booting from have the newly created SOS.DRIVER file on them. You can copy SOS.DRIVER to other disks using the File Handling Commands Menu.

If It Will Not Fit

If you attempt to generate a very large SOS.DRIVER file containing many device drivers, you might not have enough space on the System Utilities disk to create the SOS.DRIVER file. When this happens, you have two options:

1. Erase some files from the disk to open up more space.
2. Create a two-stage version of the System Utilities disk.

Two-Stage Boot

With a two-stage version of the System Utilities disk, you'll need two disks to boot the system, with each disk containing a portion of the files that need to be loaded into the Apple III's memory. To create a two-stage System Utilities disk, do the following (you might want to refer back to Chapter 3 for a review of how to use the utilities):

1. Format two disks, named UTILITY1 and UTILITY2.
2. Copy the following files from your System Utilities disk to UTILITY1:

SOS.KERNEL
SOS.INTERP
SOS.DRIVER

3. Copy the following files from your System Utilities disk to UTILITY2:

SYSTEM.MISCINFO
SYSTEM.PASCAL
SYSTEM.STARTUP

Put UTILITY1 in the built-in drive to start the system. After a few seconds, you'll see the message "Put System Disk in Built-in Drive". When you insert UTILITY2 and press RETURN, you'll see the Utilities Main Menu.

Creating a two-stage version of the System Utility disk will leave you plenty of room for the largest SOS.DRIVER you can think of.

FINAL THOUGHTS ON DEVICE DRIVERS

Remember that the file named SOS.DRIVER that's custom configured for your system *must* be on *every* disk that you use to boot the Apple III. You should go through your disks and use the System Utilities disk to delete the old versions of SOS.DRIVER and replace it with the new one.

You'll need to make up a new SOS.DRIVER using the SCP whenever

you add a new device. Although you don't have to use the SCP when you disconnect a device, if you're never going to use it again, you should either delete the driver to open up extra space on the disk or at least make it inactive so it won't be loaded into memory when you boot the Apple III.

You'll find additional details about the .PRINTER and .RS232 device drivers in Appendices G and H.

Getting Started With Business BASIC

5

Business BASIC for the Apple III is a powerful BASIC language program that's especially adapted for writing programs for a business environment. Its "core" is a BASIC *interpreter* that looks at each individual statement in a program and translates it into the 1s and 0s that the Apple III's hardware understands.

Business BASIC incorporates a number of powerful extensions and special features that make it versatile and easy to use. In this chapter, we'll cover the "basics" of Business BASIC and give you enough information so that you can start writing your own programs.

Rather than attempt to teach you how to use Business BASIC by explaining all the statements and commands, we'll ease you into the concepts of Business BASIC. In Appendix A you'll find a reference guide to all the commands and statements of Business BASIC. If you're an experienced BASIC programmer, you'll find a great deal of information there. However, if you're just getting started, we suggest you read through Chapters 5, 6, and 7 carefully. All BASICs have their own peculiarities, and attempting to get a program to run without understanding the peculiarities of Business BASIC can be a frustrating experience.

PROGRAMMING LANGUAGES

A programming language like Business BASIC allows your Apple III to perform useful work. You create a program by typing in *statements* and *commands* that tell the Apple III to manipulate data in a particular way.

You've probably heard about many different types of programming languages. BASIC (which is short for Beginner's All-purpose Symbolic Instruction Code) is a general-purpose programming language that's

designed to be easy to learn, and it does a good job of creating programs for many different applications.

There are quite a few languages available for computers, including FORTRAN, COBOL, Pascal, APL, FORTH, and C. Some (including Pascal and COBOL) are available for the Apple III. But if you're like most Apple III users, you'll find that Business BASIC is the language that will meet your needs for most applications.

GETTING STARTED

We'll assume that you've already made a backup copy of the Apple Business BASIC disk that came with your Apple III system. (If you haven't made a backup, see Chapter 2.) You'll also have to be sure that you've installed the correct device drivers on the backup disk for your system (see Chapter 4).

Starting up Business BASIC is a simple matter of putting the Business BASIC disk into the built-in floppy disk drive and turning the computer on. You'll see several messages as BASIC is loaded into the Apple III's memory. First will be the SOS (Sophisticated Operating System) message, followed by the Business BASIC message at the top of the screen. After a few seconds, "Apple III Business BASIC" will go across the screen. Finally, you'll see a catalog of the files on the disk followed by the Business BASIC prompt character ")". Anytime you see the prompt, you'll know that Business BASIC is ready and waiting for you to type in a command.

We suggest that you have Business BASIC running as you read this chapter. In that way, you can try the examples and get a solid introduction to using it.

SYNTAX

The *syntax* of a language is the set of carefully designed rules that tells you exactly how it's used. This is true of any spoken language, as well as with Business BASIC. You must type in commands and statements in exactly the right format; otherwise you'll get a "?SYNTAX ERROR" on the screen.

Some rules for using Business BASIC seem logical (like using "+" and "-" for addition and subtraction); others don't seem to make sense. Regardless, it's essential that you carefully learn the syntax for Business BASIC. It's the only way you will be able to use it.

There are three major elements to Business BASIC's syntax:

- Line numbers
- Instructions
- Data (information).

Each line of a Business BASIC program is numbered with a *line number*. Line numbers allow you to refer to the statements and commands contained on other program lines. Most Business BASIC programs contain statements that reference other program lines. For instance, commands like GOTO and GOSUB cause a program to jump to a different line.

Each line number in a Business BASIC program must be unique (you can't use the same number twice) and in the range of 0 to 63999. We'll talk about line numbers in more detail later on in this chapter.

Instructions are the statements and commands that allow you to do various tasks in Business BASIC. In most cases, a single line will contain one statement or command; however, several statements can appear in one program line.

Data is information that is used by a program. The Apple III brings data into the system (input), manipulates it in some way, and puts it out to the "real world" (output). There are a number of different types of data you'll be working with in Business BASIC. We'll cover them in the next chapter.

ESSENTIALS OF BUSINESS BASIC _____

There are several essential things that you should know to begin using Business BASIC. Since all your interaction with Business BASIC will be through the keyboard, you need to understand how to enter text. This text can be the commands used to control what happens in Business BASIC, or it can be the statements and data that make up a program.

Commands that you type can be carried out immediately in what is called *immediate mode*. One of the easiest ways to see how immediate mode works is to use the PRINT statement. Without actually knowing how to write a program, you can use the PRINT statement to display the results of calculations and use your Apple III just like you would a calculator.

Entering Text

Entering text is a simple matter of typing. If you make a mistake, use the LEFT ARROW key to move back a space and type over your mistake. If

the mistake is in the middle of a line, use the **RIGHT ARROW** key to return to where you were. If you ruin a line, press **CONTROL X** to cancel it. You can then start over from the beginning. Later, we'll talk about more advanced methods of changing (editing) text.

Using Spaces

If you're new to BASIC, you'll probably be a bit confused about where to put spaces in Business BASIC commands and lines. The best word of advice is not to worry too much about it.

Spaces help to separate the elements of a line so that Business BASIC can easily identify things like line numbers and statements. Spaces also make program listings easier to read.

It's a good idea to make a habit of putting a space between the line number and the Business BASIC command, as well as between the command and the expression that follows. (You'll see what we mean as we go along.) You should also be concerned about spaces within quotation marks because whatever is between the quotes will be printed out exactly as it appears.

Immediate and Deferred Execution

When the Apple III executes a program, it performs the actions specified by the statements you type in. There are two ways to execute a program: immediate mode and deferred mode. Execution in immediate mode occurs when you type in a line without a line number and press **RETURN**. Execution in deferred mode does not occur until you enter a separate command, **RUN**. In order to use deferred mode, all statements that you enter must be preceded by a line number. In immediate mode the Apple III looks at the line, makes sure it contains a correct BASIC statement, and does what it's directed. For instance, type

```
)PRINT "This is Business BASIC"
```

and press **RETURN**. You'll see

```
This is Business BASIC
```

If you made a mistake, for instance, typing **PRIMT** instead of **PRINT**, Business BASIC will give you a message. You'll hear a beep, and the message

```
?SYNTAX ERROR
```

will appear on the screen. This tells you that you've made a mistake.

PRINT Statement

Let's begin with the Business BASIC statement that you'll be using the most—PRINT. We've already used it in the previous examples, and you'll see it throughout the examples in this chapter. The PRINT command simply tells the Apple III to print something *on the video display*. Although it sounds a bit strange, PRINT has absolutely nothing to do with a printer. The command PRINT#, which we'll talk about in Chapter 6, is used with a printer.

PRINT has lots of power behind it, and we'll explain its fine points as we go through this chapter and the next. For now, all you need to know is that if you type PRINT followed by anything in quotation marks ("), whatever appears within the quotes will appear on the screen *exactly* as it's typed. For instance,

```
)PRINT "Hello, I'm the Apple III."
```

will appear like this when you press RETURN:

```
Hello, I'm the Apple III.
```

You can use PRINT in immediate mode as a handy (though expensive) calculator. The system will respond immediately with the answer. Try the following examples:

```
)PRINT 120+324 Addition
```

```
444
```

```
)PRINT 450-230 Subtraction
```

```
220
```

```
)PRINT 1.275*6.34 Multiplication
```

```
8.0835
```

```
)PRINT 16543/7.2 Division
```

```
2297.64
```

```
)PRINT 7.3^9 Exponentiation
```

```
5.88716E+07
```

```
)PRINT 4.557*2.12*9.05/6.31-.334^3 Combination
```

```
13.8186
```

The correct answers will appear on the line immediately following the PRINT statement. Notice that unlike our earlier examples, you don't use quotation marks. Quotation marks indicate a *literal*—everything in quotes is printed exactly the way it appears. For instance,

```
)PRINT "453.7/6.32+2.4"
453.7/6.32+2.4
```

You'll notice that you'll occasionally see results that are displayed in scientific notation. That's a short way to express very large numbers. For instance, the number

```
5.88516E+07
```

is equivalent to

```
58851600
```

The number after the "E" indicates the number of spaces you must move the decimal point to the right to get the actual number.

Let's use another example. If you type

```
)PRINT 2^40 (2 raised to the fortieth power)
1.09951E+12
```

that number is equivalent to 1099510000000.

Error Messages

As you become familiar with Business BASIC, you'll make more than a few mistakes along the way. When Business BASIC looks at something it can't do or doesn't understand, you'll hear a beep and an *error message* will appear. Error messages always start with a question mark.

Unfortunately, there are hundreds of things that you can do wrong. Business BASIC (indeed any BASIC) isn't very clever in figuring out what you mean unless it's typed in exactly the right format.

The 40 error messages that Business BASIC produces are summarized in Appendix B. As we go along, we'll point out the common error messages that you'll get in various circumstances. If you're a beginner, the most common error you'll see will be

```
?SYNTAX ERROR
```

This means you haven't typed in the command correctly. It could be as simple as typing RUM instead of RUN or it could be something more complicated. You'll soon be able to recognize common errors.

PROGRAMS

A *program* is simply a list of statements that tells your Apple III exactly what to do. A program doesn't have to be long; in fact, the one-line commands we've used up to this point are programs themselves. They do useful work, but most programs are much longer—often tens or even hundreds of lines.

Combining Statements

Most of the time, you'll want to put each individual statement on a single program line. (As you'll soon see, each line will normally have a number.) However, to save space you can put several Business BASIC statements into a single line. You do this by separating each statement with a semicolon (;). For instance, you could type the following:

```
)PRINT "Business BASIC IS VERSATILE":PRINT 2/3:PRINT "THESE ARE THE TIMES THAT
TRY MEN'S SOULS": PRINT 2/4*6:PRINT "THE QUICK BROWN FOX JUMPED OVER THE LAZY D
OG":PRINT 56^4:PRINT "THE APPLE III'S PROFILE HARD DISK CAN STORE THE EQUIVALEN
T OF 35 FLOPPIES."
```

When you press RETURN, you'll see

```
BUSINESS BASIC IS VERSATILE
.666667
THESE ARE THE TIMES THAT TRY MEN'S SOULS
3
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
9.8345E+06
THE APPLE III'S PROFILE HARD DISK CAN STORE THE EQUIVALENT OF 35 FLOPPIES.
```

Business BASIC “sees” each colon as the end of a statement and executes it. There's no limit to the number of statements that can be combined, but the total number of characters cannot be larger than 255. If you get carried away in putting statement after statement into a single line, you'll suddenly find that Business BASIC jumps back into the immediate mode, as if you typed CONTROL X.

Deferred Execution

Even though you can pack quite a few statements into a line of Business BASIC using colons, there's still only so much you can do in immediate mode. The problem with immediate mode is that you have to type in the lines every time you want to execute them.

What you need is a way to type in a program and store it away for later use—even if that is only a few minutes away.

This is where deferred execution comes in. You “defer” execution by

typing a line number before each BASIC line. As each line is typed, the Apple III stores it away in its memory, but it doesn't execute the program until you instruct it to. That's done using the RUN command.

To see how deferred execution works, type

```
)10 PRINT "This is Business BASIC"
```

and press RETURN. Nothing happens because Business BASIC has stored the line away, waiting for directions from you on when to run it. To run it, type

```
)RUN
```

and you'll see the message printed.

Editing Programs

Editing a program is also easy. You can add another line to the previous program by typing

```
)20 PRINT "For the Apple III"
```

When you type RUN, you will see the two messages on the display. If you want to change (edit) one of the lines, the simplest way is to type the line over again. For instance,

```
)20 PRINT "An advanced interpreter"
```

When you type RUN, you'll see the new line displayed.

Cursor Move Mode

Although retyping the whole line is the simplest way to correct a mistake, if you've entered a long and involved line, the last thing you want to do is type the whole line over again. Fortunately, there's another way to edit program lines that lets you change only what you need to. It's called *cursor move mode*.

To enter the cursor move mode, press ESCAPE after you've entered a line. A plus sign (+) will appear in the cursor. This indicates that you can move the cursor through the text without changing anything.

For example, if you've typed in the line

```
)10 PRINT "I LOVE MY APPLE III"
```

instead of typing the line over because you spelled PRINT incorrectly,

just press ESCAPE. When you see the plus sign in the cursor, use the arrow keys to move the cursor to the *beginning* of the line. When you enter cursor move mode, you must move the cursor to the beginning of the line you want to correct.

At this point, you must *exit* the cursor move mode by pressing ESCAPE again. Use the RIGHT ARROW key to move the cursor to the first incorrect character (the "M" in PRINT). What you've just done is "read in" the characters up to that point.

Type N to correct the incorrect character and then use the RIGHT ARROW key to move the cursor to the *end* of the line. When you press RETURN, the new line will be read by the Apple III.

The cursor move mode can save you time when you type in a long line and find that it's incorrect. Give it a try with a few more lines to see how it works.

More About Line Numbers

By convention, program lines are usually numbered in increments of ten. You can use any increment you wish, but using increments of at least ten gives you the opportunity to add additional lines in the program without having to retype the program.

Let's look at how this works. Enter the following program:

```
)10 REM A Quick Program
)20 PRINT "This is a line of text"
)30 END
)RUN
This is a line of text
```

REM stands for REMARK. Anything you type after REM is ignored but it's handy for making notes to yourself about the program.

If you wanted to add another line of text to your program, it would be a simple task. Since Business BASIC executes line numbers in a sequential manner, all you need to type is

```
)25 PRINT "This is another line of text"
)RUN
This is a line of text
This is another line of text
```

As you see, the new line was automatically added to the program. It's also important to remember that no two lines can have the same number. If you inadvertently type in two lines with the same line

number, only the second one you typed in will be stored. Try the following:

```
)NEW
)10 REM Another short program
)20 PRINT "This is text"
)20 PRINT "This is more text"
)30 END
)RUN
This is more text
```

As you see, the first line 20 was ignored by Business BASIC. In the same way, you can change a program line (it's called editing the program) simply by typing the line number followed by the new program line.

Looking at the Program in Memory

You can type **LIST** at any point to see a list of the entire program you've stored in memory. It doesn't matter in what sequence line numbers were entered because you'll see the listing in numerical order. Assuming you haven't erased the previous program, typing **LIST** should give you the following:

```
)LIST
    10  REM Another short program
    20  PRINT "This is more text"
    30  END
)
```

Notice that the listing looks a little different from what you typed; there are some extra spaces between the line numbers and the beginning of commands. That's because Business BASIC automatically *formats* the listing. We'll talk more about that in a while, but for now, remember that as you enter longer and longer programs, certain lines are automatically indented to make things easier to read.

If the program is longer than 24 lines (and many of them are), it will scroll up over the top of the screen. You can make the listing pause anywhere by typing **CONTROL 7**. Note that only the 7 key on the numeric keypad can be used to do this. Typing **CONTROL 7** again will resume the listing. **CONTROL C** will stop the listing and return you to the Business BASIC prompt character.

There are several ways to use **LIST** in addition to listing the entire program. You can list parts of a program. For instance,

)LIST 50

will list only line 50 of your program.

)LIST 50 TO

will list from line 50 to the end of the program.

)LIST TO 50

lists from the beginning of the program to line 50.

There are several other ways to list parts of programs:

)LIST 50 TO 100

)LIST 50-100

)LIST 50,100

will list all the lines between 50 and 100.

Erasing a Program

Unless you save your program (we'll discuss that later), it will be erased every time you turn the Apple III off. But you don't have to do that to erase it; just type **NEW**. Everything you've stored will be erased from memory and the Apple III will be ready for a new program.

Erasing Parts of Programs

There will be times when you don't want to erase your entire program. Instead, you'll want to erase a line or several lines. This is done using the **DEL** command. It's short for "delete," and it lets you selectively erase program lines. It can be a single line

)DEL 20

or a range of lines

)DEL 20 TO 100

The last form deletes all lines between 20 and 100 inclusive. There are several ways to delete a range. You could also type

)DEL 20-100

or

```
)DEL 20,100
```

Both delete the entire range of lines.

There's also a short way to erase a single line. Just type the line number, followed by RETURN. Business BASIC sees this as a "null line" and ignores it.

WORKING WITH MEMORY

The Apple III's RAM (random-access memory) stores the program you type in until you erase it, save it to disk, or turn off the computer. There are two commands you can use to manage the Apple III's memory: NEW and FRE.

NEW

When you are ready to start typing in a new program or are about to load a program from disk, you need to clear the previous contents of the Apple III's program memory. To do this you use the NEW command.

The NEW command clears any programs and data that are stored in the area of memory reserved for BASIC programs and data.

FRE

FRE tells you how much space you have left in memory. For instance, if you have a 256K system, type

```
)PRINT FRE
```

Depending on what device drivers you've installed on your Business BASIC disk and the number of programs you've stored away, you'll get a number somewhere around 196,000. The rest of the memory is used to store the Apple III's *system software*—the SOS and the BASIC *interpreter*.

FRE is handy for finding out how large a program is. If you type FRE before and after loading a program into memory, you can determine by subtraction how much memory the program uses.

SAVING AND GETTING PROGRAMS

Although it's fine to type a program into memory, you won't want to type in the same program every time you turn on your computer. Pro-

grams are saved as *files* on a disk—either a floppy disk or the Profile hard disk. You can do a great deal with files in Business BASIC. Most of the advanced features will be covered in the next two chapters. For now, there are a few commands you should learn to allow you to save and load program files stored on disk.

SAVE

To save a program you've just typed, type **SAVE** and a *pathname* (see Chapter 3). Here are a few examples:

```
)SAVE .D1/Invcount
)SAVE .PROFILE/Checkbal
```

The first example will save a basic program called “Invcount” on a floppy disk. The second saves “Checkbal” on the Profile hard disk.

Naming Files

In Business BASIC, the rules for naming a program are the same as those for naming a file, as we discussed in Chapter 3. To review, a *local name* (the name of the program you're saving) must begin with a letter and can be as many as 15 characters in length. It can contain letters, numbers, or periods—but no spaces or other punctuation. Here are a few examples:

```
QUICKPRINT.DEMO
TICTACTOE
LISTSORT
```

Although you can use any name you want, by convention most BASIC programs are named as a single string of characters.

Note that even if you type in lowercase characters, the Business BASIC program name will be stored as all uppercase characters.

Be careful that you don't use a name that already exists on the disk you're putting the file on. If another program has the same name, it will be erased.

CAT

The CAT (catalog) command gives you a list of the files that are stored on a device. If you just type **CAT**, you'll get a list of the files stored on the disk in the Apple III's built-in disk drive. Typing **CAT** *pathname* will show you the list of files on the device or subdirectory you specify. For instance,

```
)CAT .Profile
```

will show you a list of the files on the Profile hard disk. We'll talk more about CAT in Chapter 7. For now, remember that all files of type "BASIC" are Business BASIC programs that you can run.

LOAD

Load takes a Business BASIC program from a disk and puts it into the Apple III's memory. To load an existing program, type **LOAD** and the *pathname*. Here are two examples:

```
)LOAD .D2/Expense
```

```
)LOAD .PROFILE/Income
```

If the file you've specified isn't on the disk you've specified, you'll get the message

```
?FILE NOT FOUND ERROR
```

If you try to load a file that isn't written in Business BASIC, you'll see the message

```
?TYPE MISMATCH ERROR
```

STARTING AND STOPPING PROGRAMS

Once you have entered or loaded a program into memory, you will want to execute it. The commands **RUN** and **CONT** can be used in immediate mode to get a program running. Including the statements **STOP** and **END** in a program will allow you to determine where your program will stop or end.

RUN

The **RUN** command is Business BASIC's "starting gun." When you type it in, the Apple III begins to execute the program that's in memory. It begins by executing the line with the lowest line number.

If for some reason you want to start the program at a particular line number (you might want to do it for "debugging") you can type **RUN** and a line number, such as

```
)RUN 100
```

You can also run a program that isn't loaded into memory yet by specifying a *pathname*. For instance,

```
)RUN .PROFILE/INCOME
```

will load and run a hypothetical program named "Income" from the hard disk.

STOP

STOP is used as a statement within a program to halt execution immediately. When a STOP is executed, the line where the program was stopped is displayed. For instance, a STOP statement in line 320 would result in

```
?BREAK IN 320
```

Remember that you can only use STOP in deferred mode. You can't type it in from the keyboard to stop a program.

CONT

CONT is used in immediate mode. You can start up a program halted by STOP with CONT (for continue). The program will then start from where it left off.

END

END is a statement used within a program that works exactly like STOP, except that it doesn't display a message stating where the program stopped. It's good programming practice to use END as the last program line of a Business BASIC program.

CONTROL KEYS

There are a number of special keys (and combinations of keys) that can be used to control what's happening while you are running a program.

CONTROL C

Holding down the CONTROL key and typing c while a program is running will stop the program immediately. The system will then return to

immediate mode. CONTROL C is just like the STOP statement explained earlier, except that you can't put it in a program.

However, remember that the program is still in memory. Consequently, you can start the program running from where it stopped by using the CONT command.

There are several instances when CONTROL C will *not* stop a program. One instance is when you've used an ON KBD command in your program. If the program is waiting for a key to be pressed, it will see CONTROL C as just another key. Also, if the program is waiting for an input/output (I/O) operation to finish transferring a disk file or it is printing, Business BASIC will ignore a CONTROL C. (These advanced concepts will be explained in Chapter 6 and Chapter 7.)

RESET

Pressing RESET is exactly like pressing CONTROL C. Make sure you don't press CONTROL and RESET at the same time. That causes a complete system reset, and anything that's in memory is erased.

Other CONTROL Keys

Business BASIC has a number of special features that are initiated by holding down the CONTROL key and typing a number from the *numeric keypad* only. Admittedly, you probably won't need to use these very often, but you should know about them.

CONTROL 5 This feature allows programs to run faster. When you type it, the screen will go blank. Since the Apple III doesn't have to use processor time and memory to keep characters on the screen, a large program will run faster. How fast it runs depends on the program. To get the screen image back, just type CONTROL 5 once again.

CONTROL 6 Earlier in this book, we talked about the Apple III's typeahead buffer. It's a place in memory where the commands you type in from the keyboard are stored until the computer is ready to act on them. For instance, you might want to type in a command for the Apple III to use as soon as it's finished working with a current program. If you make a mistake, you can clear all the characters you've typed in out of the typeahead buffer by typing CONTROL 6. (It won't affect the program that's currently running.)

- CONTROL 7** When you type CONTROL 7, all output to the screen is suspended. Typing another CONTROL 7 will allow the screen output to continue. You use this command to prevent lengthy displays from scrolling off the screen before you have a chance to read them.
- CONTROL 8** This command makes the Apple III ignore the special-purpose screen control codes. See Appendix E for a list of these codes.
- CONTROL 9** This command temporarily prevents any program output from being displayed on the screen; however, the program will continue to run (as opposed to CONTROL 7 which stops the program). Any output sent to the screen while the screen is disabled is lost. Typing another CONTROL 9 will allow the output to once again be displayed.

Programming In Business BASIC

6

Now that we've looked at getting started with Business BASIC, it's time to fill in the details. This chapter explains Business BASIC concepts and shows you ways to combine the various statements and commands into useful programs. If you're new to BASIC programming, pay attention to this chapter. Even if you've done BASIC programming before, we suggest that you read this chapter. Not only is it a review, but also it's a starting point for the features that are unique to Apple III Business BASIC.

CONSTANTS

A *constant* is a value that never changes. Business BASIC has two kinds of constants:

- A *numeric* constant is a number. Examples might be PI (3.1416) or the number of feet in a mile (5280).
- A *string* constant is anything you enclose in quotation marks (like after a PRINT statement).

VARIABLES

You can think of a variable as a place where a value is stored. To clear up the concept, you can think of it as a container. You give each variable a name. In Business BASIC, a variable name always starts with a letter and can be up to 64 characters long. You can use letters, numbers, or

periods in it. However, you can't use any spaces or other punctuation. Here are a few examples:

NUMBEROFPEOPLE

CHECKS0183

INVENTORY.WIDGETS

Those are legal variable names. Here are a few illegal ones:

NUMBER OF PEOPLE (Contains spaces)

0183CHECKS (Begins with a number)

INVENTORY,WIDGETS (Contains a comma)

Types of Variables

Business BASIC has four types of variables; each is used to store a specific type of data. The four types of variables are

- Real
- Integer
- Long integer
- String.

You tell Business BASIC which type of variable you're using by selecting one of the following characters as the last character of the variable name:

No character	= Real
%	= Integer
&	= Long integer
\$	= String

Here are some examples of variable names and their types:

Income	Real
Employees%	Integer
Stars&	Long integer
Coname\$	String

Business BASIC creates a variable the first time a variable name is used in a program. Every time Business BASIC comes across a variable name in a program, it first checks to see whether it already exists. If

Standard Notation	Scientific Notation
1000000000	1E09
.000000001	1E-09
300	3E2
-1234567890	-1.23457E09
-.00000123456789	-1.234E-06

You should note that in Business BASIC, any number whose absolute value is smaller than $2.9388E-39$ is automatically converted to 0.

Business BASIC automatically displays real numbers as six digits. Longer numbers are rounded off. Leading zeroes to the left of the decimal point and trailing zeroes to the right aren't shown.

Integers

An integer is any positive or negative number without a decimal point or fractional part. A number without a sign is assumed to be positive. In Business BASIC, the range of integers is -32768 to 32767 . Here are a few examples:

```
0
-234
675
-30765
31654
```

Using integers whenever you can speeds up calculations in Business BASIC. Never use a number like 7655.0 (a real number) when you could use 7655 instead.

Remember that in Business BASIC, a variable name for an integer must end with a percent sign (%).

Long Integers

A long integer is an integer that is as many as 19 digits long. The range for long integers is -9223372036854775808 to 9223372036854775807 . A variable name for a long integer must end with an ampersand (&). Here are a few examples of legal long integers:

```
-3456743268787665154
12
983
-876543
0
```

As you can see, the range of long integers overlaps the range of integers.

When would you use long integers? Obviously, scientific work often requires numbers this large. It's worth noting that the Apple III works on integers much faster than on real numbers. Therefore, you can use long integers to speed up calculations in programs that use dollars and cents. You can do this by changing those dollars and cents into pennies. For example \$12,231.67 would be 1223167 pennies. Keep this in mind as you write your own programs.

STRINGS

A string is a sequence of up to 255 characters enclosed in quotation marks. (A string with 0 characters is called a null string.) The name of a string variable must end with a dollar sign. We've already used a few strings in the last chapter, but here are a few more examples:

```
"MAN IS BORN TO SUFFER AND DIE"
```

```
"ACCOUNT #320-67"
```

```
"DECEMBER 21, 1984"
```

```
"GEORGE JONES"
```

```
"AMALGAMATED WIDGETS, INC."
```

Within a string you can use any characters the Apple III keyboard can generate except quotation marks. You can use single quotes in a string. For instance,

```
"Type a "RETURN" when you see the prompt"
```

is not a legal string.

```
"Type a 'RETURN' when you see the prompt"
```

is legal.

RESERVED WORDS

All the words that have a special meaning in Business BASIC are called *reserved words*. In Appendix C, you'll find a list of all of Business BASIC's reserved words. You'll get error messages if you use any of these words for variable names.

It's perfectly all right to use reserved words in a string (surrounded by quotes), since a string isn't considered a command or a variable.

ARRAYS

An *array* is a way of naming an ordered collection of individual variables. Arrays are used extensively in many Business BASIC programs and are an important concept to understand.

Arrays can simplify programming by giving a large collection of similar items a single *array name* instead of naming each item individually. Suppose you had a table of several hundred numbers. To say the least, it's much simpler to give them a single name than to make up several hundred different names. (It also saves memory space.) You can then identify individual numbers by their location in the table.

Each variable within an array is called an *element*. In keeping with the usual way of doing things in computers, the elements in an array are numbered starting with 0.

For instance, if you wanted to have Business BASIC display the value of the 23rd element of an array called BOXES, you would type

```
)PRINT BOXES(23)
```

The number in parentheses that identifies the individual element within an array is called the *subscript*.

Here's an example of a simple array:

Rita	George	Carl	Chris	Carla	Dave	Henry	Nancy	Joseph
NAME\$(0)	NAME\$(1)	NAME\$(2)	NAME\$(3)	NAME\$(4)	NAME\$(5)	NAME\$(6)	NAME\$(7)	NAME\$(8)

This string array, NAME\$, is made up of nine elements, each of which is a name. To display the third element you can enter

```
)PRINT NAME$(2)
)CARL
```

Before we go any further, remember that in the case of large arrays you must specify the number of elements in an array before you can create one. This is done using a DIM statement, which we'll talk about later.

Array Dimensions

An array can have more than one *dimension*, meaning that more than one subscript is required to select an individual element.

The simplest array, a one-dimensional array, is simply a list of variables, like the previous example. A two-dimensional array is organized like a table with rows and columns. For instance, here's a two-dimensional integer array called WIDGETS%:

Widgets%(0,0)	Widgets%(0,1)	Widgets%(0,2)	Widgets%(0,3)	Widgets%(0,4)	Widgets%(0,5)	Widgets%(0,6)	Widgets%(0,7)	Widgets%(0,8)
Widgets%(1,0)	Widgets%(1,1)	Widgets%(1,2)	Widgets%(1,3)	Widgets%(1,4)	Widgets%(1,5)	Widgets%(1,6)	Widgets%(1,7)	Widgets%(1,8)
Widgets%(2,0)	Widgets%(2,1)	Widgets%(2,2)	Widgets%(2,3)	Widgets%(2,4)	Widgets%(2,5)	Widgets%(2,6)	Widgets%(2,7)	Widgets%(2,8)
Widgets%(3,0)	Widgets%(3,1)	Widgets%(3,2)	Widgets%(3,3)	Widgets%(3,4)	Widgets%(3,5)	Widgets%(3,6)	Widgets%(3,7)	Widgets%(3,8)
Widgets%(4,0)	Widgets%(4,1)	Widgets%(4,2)	Widgets%(4,3)	Widgets%(4,4)	Widgets%(4,5)	Widgets%(4,6)	Widgets%(4,7)	Widgets%(4,8)

To identify the fourth element in the fifth row of this array you would type

```
)PRINT WIDGETS%(4,3)
```

There are also three-dimensional arrays. One example is an array to describe the popular cube puzzle. You need three subscripts to identify the row, column, and layer that an element is positioned in.

But arrays can have four or more dimensions. A four-dimensional array is a bit difficult to visualize, but it is possible. In most cases you won't need an array of more than three dimensions.

Array Contents

An array can contain only one type of variable—real, integer, long integer, or string. The same naming conventions we talked about earlier regarding variables apply to array names. Here are a few examples:

SALES	Real
ACCOUNT%	Integer
PENNIES&	Long integer
CONAME\$	String

Dimensioning Arrays

As mentioned earlier, if you want to create a large array, you tell Business BASIC how much space to allocate to the array. That's done

using the DIM statement. If you don't use DIM, Business BASIC automatically creates an array having 11 elements (0-10) in each dimension. For instance,

```

)DIM Names$(39,29,4)
    
```

creates a three-dimensional string array having 40 elements (0-39) down, 30 elements (0-29) across, and 5 (0-4) levels. That's a total of 4000 "boxes" ($40 \times 30 \times 5$) to store strings in (see Figure 6-1).

NOTE: Since array numbering begins with 0, the number in a DIM statement is actually one integer smaller than the size of the array you want to create. The maximum number of elements in each dimension of an array is 32767.

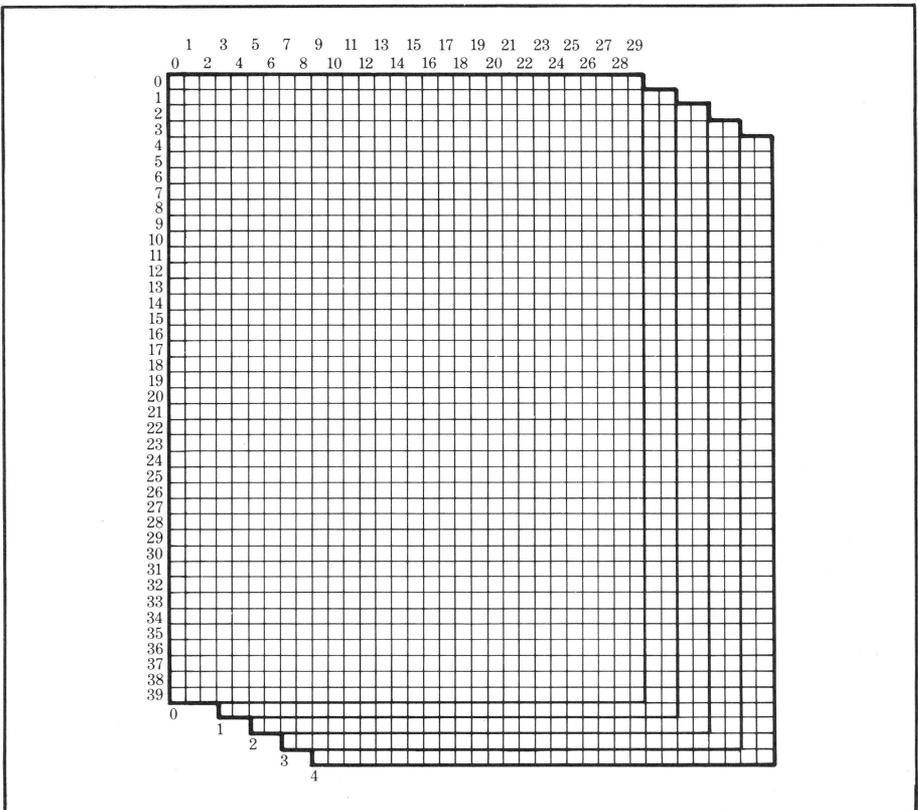


Figure 6-1. Representation of a 40x30x5 array

Table 6-1. Arithmetic Operators

Sign	Meaning	Example
+	Plus sign	+43
-	Minus sign	-254
+	Addition	64+32=96
-	Subtraction	42-20=22
^	Exponentiation	3^6=729
*	Multiplication	7*11=77
/	Division	12/1.3=9.23077
MOD	Modulo (Long integers only)	7 MOD 5=2
DIV	Integer Division (Long integers only)	8 DIV 5=1

EXPRESSIONS

Business BASIC has five types of expressions:

- Arithmetic
- Long integer
- String
- Relational
- Boolean.

An *expression* can be as simple as a single variable or constant, or it can be a long mathematical expression that includes *arithmetic operators* (symbols of mathematical calculations) and *operands* (the numbers that are operated on). There are also string operators (to manipulate strings), relational operators (to compare two values), and Boolean operators (to perform the logical operations AND, OR, and NOT).

Business BASIC has nine arithmetic operators. They are given in Table 6-1.

If you're not familiar with integer arithmetic, *modulo* is the integer remainder of a division, and integer division is the integer result of a division operation.

Precedence of Operators in Expressions

An expression often calls for more than one operation to occur. For example

```
)PRINT 5*6/1.5^2
13.3333
```

calls for multiplication, division, and exponentiation. *Precedence* is the order in which arithmetic expressions are evaluated. In Business BASIC, the order in which arithmetic operations are performed is as follows:

1. Any statements enclosed in parentheses are evaluated.
2. Plus or minus signs are applied to numbers.
3. Exponentiation.
4. Multiplication, division, MOD, and DIV.
5. Addition and subtraction from left to right.

You can change the order in which Business BASIC evaluates arithmetic expressions by using parentheses. If more than one set of parentheses is included in an expression, they are evaluated from left to right.

Nesting is when one set of parentheses is included within another set. In this case, Business BASIC evaluates the expression from the innermost set outward. You can use as many sets of parentheses as you wish. Don't be afraid to use them often to clarify the way you want expressions evaluated.

You need to be careful, however. Parentheses can vastly change the results of an expression. Changing the previous example to

```
)PRINT (5*6/1.5)^2
400
```

gives a result that is considerably different.

Relational Expressions

Business BASIC's relational operators let you compare two values. There are six relational operators:

Operator	Operation
=	Equal to
<	Less than
>	Greater than
<= or =<	Less than or equal to
>= or =>	Greater than or equal to
<> or ><	Not equal to

All relational operators have the same precedence and are evaluated from left to right.

Business BASIC arbitrarily assigns 0 to a false condition and 1 to a true condition. Here are a few examples:

34 > 8	Results in 1 (true)
34 > 213	Results in 0 (false)
123 >= 123	Results in 1 (true)
"Bill" > "Bill"	Results in 0 (false)
"Gil" >< "Cal"	Results in 1 (true)

You can include relational expressions within other expressions as in the following examples:

345 + (23 <> 23)	is the same as	345 + 0
(X + 1) * (56 > 34)	is the same as	(X + 1) * 1

Comparing Strings

There are two rules that Business BASIC uses when comparing strings. First, two strings that aren't the same length aren't equal. Second, if the first string is identical to the the first part of a longer string, the longer string is greater than the first string. Characters in strings of equal length must be exactly the same in order for the strings to be equal. Strings are compared one character at a time from left to right.

Boolean Expressions

Because Business BASIC's Boolean operators give you the ability to make logical decisions they're often called *logical operators*. There are three standard Boolean operators in Business BASIC:

AND
OR
NOT

Boolean logic can be a bit confusing at first, but as you'll see, it's very logical.

Let's use a few quick examples. Imagine you and a friend have extensive book collections, which we'll call Library1 and Library2.

- Library1 AND Library2 = The list of books that both you and your friend have in common.

- Library1 *OR* Library2 = All the books in both collections.
- Library2 *NOT* Library1 = The books in Library2 that are NOT in Library1.

That's all well and good, but your Apple III works with numbers, not books. In Business BASIC, a Boolean operation is performed using the binary representation of a number. Since relational expressions evaluate to either 1 or 0, Boolean operators are often used to combine several relational expressions.

The following tables summarize how Boolean expressions are evaluated in Business BASIC. These tables are called *truth tables*.

AND results in 1 only if *both* values are 1.

1 AND 1 = 1

1 AND 0 = 0

0 AND 1 = 0

0 AND 0 = 0

OR results in 1 if *either* value is 1.

1 OR 1 = 1

1 OR 0 = 1

0 OR 1 = 1

0 OR 0 = 0

NOT logically complements each value.

NOT 1 = 0

NOT 0 = 1

If more than one Boolean operator is included in an expression, the Boolean operators are evaluated from left to right. Here are a few examples of Boolean expressions:

NOT (6+2>=5) Results in 0 (false)

NOT ("Apple" <> "Apple") Results in 1 (true)

("Basic" = "Business BASIC") OR ("Apple" = "Apple")
Results in 1 (true)

("Basic" = "Business BASIC") AND ("Apple" = "Apple")
Results in 0 (false)

ASSIGNMENT STATEMENTS

Assignment statements are one of the most common things in Business BASIC programs. They assign the values to variables. Here's an example:

```
)A=345.89
```

The variable to the left of the equal sign is *assigned* the value of the expression on the right. You could also type

```
)LET A=345.89
```

LET is a reserved word that's optional in an assignment statement. Most people leave it out.

Here are a few examples of assignment statements:

```
)LENGTH=1256.21
)A$="Apple III Business BASIC"
)HOUSES%=2000
)SDF&=2.38E2
```

Assigning values to elements of an array looks something like this:

```
10 FRUIT$(1)="APPLE"
20 FRUIT$(2)="ORANGE"
30 FRUIT$(3)="BANANA"
40 FRUIT$(4)="KIWI"
```

(A string variable name must end with a "\$" and the string must be in quotes.)

You can also type multiple assignment statements on a single line. For example:

```
10 FRUIT$(1)="APPLE":FRUIT$(2)="ORANGE":FRUIT$(3)="BANANA":FRUIT$(4)="KIWI"
```

Assignment statements can also contain expressions like

```
Z=(X*Y^2)
```

SWAP

SWAP is a Business BASIC statement that swaps the values of two variables. It works like this:

```
)X=20
)Y=40
)PRINT X:PRINT Y
```

```

20
40
)SWAP X,Y
)PRINT X:PRINT Y
40
20

```

It goes without saying that the variables you swap must be the same type. You can't, for instance, swap a string and an integer.

Clearing Variables

Business BASIC's CLEAR statement is similar to NEW, which we discussed earlier. There is, however, one important difference: CLEAR doesn't erase the program that's currently in the Apple III's memory. Instead, it initializes all the variables in the current program by setting all numeric variables and numeric array elements to zero, and every string variable and string array element to the null string (a string of length zero).

Most of the time, you'll use CLEAR when you're in immediate mode, but it can also be used from within a program.

DATA AND READ STATEMENTS

There's another way to assign values to a list of variables. Look at the following example:

```

10 DATA 10,24,63,934
20 READ W,X,Y,Z

```

Each value in the DATA statement is read into the corresponding variable in the READ statement. After this program is run, W=10, X=24, Y=63, and Z=934.

Obviously, the number of elements in the DATA and READ statements must be identical. If you have too many variables and not enough data, you'll get an "?OUT OF DATA" error. If you have too many individual pieces of data and not enough variables, the extra data will be ignored. Since you won't get an error message in this case, it's a difficult problem to find.

You can mix data types in a READ statement, but the variable names must have the correct suffix. For example:

```

10 DATA 23,"APPLE III",763.54
20 READ PERSONNEL%,COMPUTER$,SALES

```

RESTORE

RESTORE allows you to read the same data more than once by restoring the "pointer" that keeps track of data back to the beginning of the list. RESTORE is used as follows:

```

10 DATA "MARC", "RALPH", "KEVIN"
20 READ EMPLOYEE1$, EMPLOYEE2$, EMPLOYEE3$
30 REM READ SAME DATA INTO DIFFERENT VARIABLES
40 RESTORE
50 READ AUTHOR1$, AUTHOR2$, AUTHOR3$

```

If you try to read the same data without RESTORE, you'll get an error message.

INPUTTING DATA FROM THE KEYBOARD

Up to this point, we've talked about how a program gets the data it needs to work on from *within* a program. However, there will be many occasions when you will want a program to *ask* the person who's using it to enter the needed data from the keyboard. For this, you'll use INPUT and GET.

INPUT

An INPUT statement tells the computer to wait for the user (you or another) to type something in from the keyboard. Until something is typed, nothing else happens. It's often used in conjunction with a PRINT statement that prints a message on the video display. For example:

```

10 PRINT "Enter a whole number"
20 INPUT NMBR%
)RUN
Enter a whole number
?

```

The question mark indicates that the Apple III is waiting for input. When you enter a number and press RETURN, the variable NMBR% is assigned the value you typed in.

You can also combine a message with the INPUT statement as follows:

```
10 INPUT "Enter a whole number";NMBR%
)RUN
Enter a whole number
```

In this case, a question mark *doesn't* appear.

GET

GET is a special variation of INPUT. Its job is to assign a single character or number *from the keyboard* to a variable in your program. Depending on the variable type you specify, the entry is treated as a string (letter or punctuation) or a number. For instance,

```
10 GET CHARACTER%
20 PRINT CHARACTER%
30 REM If entry is 8 then end program
40 IF CHARACTER%8=8 THEN END ELSE GOTO 10
```

This program patiently waits for you to type an 8 on the keyboard and stops the program when you do. The important things to remember about GET are that it can only be used for a *single* character or number and you don't press RETURN following the character—just press a key.

BRANCHING

In Business BASIC, statements are executed in ascending order. But there will be times when you'll want to jump to other sections of the program. That's called *branching*. There are two kinds of branching: *unconditional* and *conditional*.

An unconditional branch occurs when Business BASIC branches regardless of whatever else is in the statement. It's done with the GOTO statement. Conditional branching lets the program make choices depending on the data. It's done with IF-THEN, IF-GOTO, and IF-THEN-ELSE statements.

WARNING: Too much branching can turn an otherwise readable program into a mess. Use branching carefully and only when you need to.

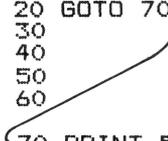
GOTO Statement

GOTO is a simple branch statement that lets you specify where the program should go next. For instance:

```

10 X=5
20 GOTO 70
30
40
50
60
70 PRINT 5
80

```



In this program, the sequence of execution will be 10, 20, 70, 80. Remember that you would need another statement to branch back to line 30; otherwise, it would never get executed.

If you attempt to branch to a line that doesn't exist, you'll get a "?UNDEF'D STATEMENT" error.

GOTO is a handy tool for executing the same group of statements many times. It's certainly easier than typing in the same thing repeatedly. Here's a program that will keep on running until you press RESET or CONTROL C. A loop like this is called an *infinite loop*.

```

)10 REM An Infinite Loop
)20 PRINT "The Apple III is a smart computer"
)30 GOTO 20
)RUN
The Apple III is a smart computer
.
.
.

```

Conditional Branching

Conditional branching is one of the keys to the power of Business BASIC. It allows the computer to choose where to go depending on the value of a variable.

Business BASIC has three conditional branching statements:

IF-THEN

IF-THEN-ELSE

IF-GOTO

IF-THEN and IF-GOTO are similar in that they specify what a program will do depending upon the value of a logical expression. The

form of the IF-THEN and IF-GOTO statements is

IF expression THEN expression or linenumber

IF expression GOTO linenumber

If the expression is true, then the expression or statement specified by *linenumber* is executed. Most of the time, relational and Boolean expressions are used, although arithmetic expressions can also be used. Here are some examples:

```
IF X=C-5 THEN PRINT Error$
```

```
IF C$<>"Receipts" THEN Calc=0
```

```
IF X<=23 AND Y<56 THEN 200
```

IF-THEN allows you to specify either a line number or another expression. IF-GOTO only lets you specify a line number. Here are three statements that are equivalent:

```
IF X=5 THEN 80
```

```
IF X=5 GOTO 80
```

```
IF X=5 THEN GOTO 80
```

IF-THEN-ELSE

With the IF-THEN and IF-GOTO statements, the program just proceeds to the next line number if the expression is not true. IF-THEN-ELSE allows you to specify an alternate operation to perform if the expression isn't true. The form for the IF-THEN-ELSE statement is

IF expression THEN statement or linenumber :ELSE statement or linenumber

Here are a few examples of IF-THEN-ELSE statements.

```
IF X=C-5 THEN PRINT ERROR$:ELSE 160
```

```
IF C$<>"RECEIPTS" THEN CALC=0:ELSE CALC=1
```

```
IF X<=23 AND Y<56 THEN 200:ELSE PRINT A$
```

It's important to note that if you use an ELSE statement without IF-THEN, Business BASIC will see it exactly like a REM statement. The following statements are equivalent:

```
REM A Check-Balancing Program
```

```
ELSE A Check-Balancing Program
```

FOR AND NEXT STATEMENTS

The easiest way to create a loop is to use the FOR and NEXT statements. Instead of the infinite loop we created earlier, let's use a FOR-NEXT loop to display the message a number of times.

```

)10 REM A FOR-NEXT Loop
)20 FOR X=1 TO 5
)30 PRINT "The Apple III is a smart computer"
)40 NEXT X
)RUN
The Apple III is a smart computer

```

As you can see, the statement between the FOR and NEXT statements is executed the number of times specified by the *index variable*, which appears after the FOR.

Here is another example: a program that will fill an array with all integers from 0 to 999 and print the results:

```

10 DIM Numbers%(999)
20 FOR X=0 TO 999
30   Numbers%(X)=X
40   PRINT Numbers%(X)
50 NEXT X
60 END

```

When you run this program, you'll see all the numbers from 0 to 999 displayed on the screen.

STEP

The index variable can be incremented or decremented by steps using the STEP option with the FOR statement. To illustrate, type **NEW** to clear the Apple III's memory and then enter the following program:

```

)10 FOR X=0 TO 40 STEP 5
)20 PRINT X
)30 NEXT X
)RUN
0
5
10
15
20

```

```

25
30
35
40

```

X is incremented by the value of the step (in this case 5). The step value can be any integer. If you don't use STEP, Business BASIC assumes a value of 1.

You can also use steps to decrement the index variable:

```

)NEW
)10 FOR X=99 TO 66 STEP -3
)20 PRINT X
)30 NEXT X
)RUN
99
96
93
90
87
84
81
78
75
72
69
66

```

Using a negative number with STEP is a good way to establish a "countdown."

Nested Loops

The FOR-NEXT loop is one of the most common statements you'll find in Business BASIC programs. In fact, FOR-NEXT loops are used so much that they're often *nested* one inside the other. Here's an example:

```

)NEW
)10 FOR X=1 TO 2
)20 PRINT X-1
)30 FOR Y=3 TO 6
)40 PRINT X+Y
)50 NEXT Y
)60 NEXT X
)RUN
0
4
5
6

```

```

7
1
5
6
7
8

```

Although it looks complicated at first, nested loops like this one aren't really that difficult to understand. It will help to LIST the program since Business BASIC automatically indents FOR-NEXT loops—making them much easier to read.

```

)LIST
10  FOR X=1 TO 2
20  PRINT X-1
30  FOR Y=3 TO 6
40  PRINT X+Y
50  NEXT Y
60  NEXT X

```

To help you further understand how this nested loop works, type **TRACE** and run the program. As it runs, you'll see the line numbers of the program listed as they're executed. Since putting arrows on the listing would make it difficult to read, here's the order in which this program is executed and the results:

```

#10 #20 0
#30 #40 4
#50 #40 5
#50 #40 6
#50 #40 7
#50 #60 #20 1
#30 #40 5
#50 #40 6
#50 #40 7
#50 #40 8
#50 #60

```

Spend a few minutes studying how this nested loop works. It's really not that complicated if you remember that the innermost loop runs the number of times you specify before returning to the outer loop.

A very common error that can be hard to trace occurs when the outer loop ends before the inner loop. For instance in the previous program,

```

50 NEXT X
60 NEXT Y

```

would give you incorrect results and an error message.

You can nest as many as nine levels of FOR-NEXT loops, although keeping track of that many would be a chore. Nesting more than nine levels results in a "STACK OVERFLOW" error.

SUBROUTINES

As you get experience in Business BASIC programming, you'll soon find that there are quite a few operations that you'll be doing repeatedly in the same program. A *subroutine* allows you to reexecute a section of your program to repeat a task.

You might think of a subroutine as a miniature program at the end of a longer "main" program. The main program will branch to the subroutine, and once the subroutine has been executed, the main program will resume at the point where it left off. In Business BASIC, you use a GOSUB statement to tell the program to "go to" a subroutine and a RETURN statement to return to the main part of the program.

Let's use a simple example:

```

10  REM An Example of GOSUB
20  INPUT"ENTER NUMBER";X
30  GOSUB 100
40  PRINT"THE ANSWER IS ";Y
50  END
100 Y=100*X/23
110 RETURN

```

This example is so simple that you probably wouldn't use a subroutine for it, but you can see that as soon as you enter the number, the program branches to the subroutine, performs the needed calculations, and returns to the main program where the results are printed on the screen.

Now that you see how a subroutine works, let's use a more practical example. Here's a program with a subroutine that delays printing while it counts through a FOR-NEXT loop.

```

10  REM A Time Delay Display
20  PRINT"This is the Apple III"
30  GOSUB 300
40  PRINT"A versatile computer system"
50  GOSUB 300
60  PRINT"Especially with Business BASIC"
70  GOSUB 300
80  PRINT"And the Profile Hard Disk"
90  END
300 REM Time Delay Subroutine
310 PRINT"Wait a moment!"
320 FOR X=1 TO 3000
330   NEXT X
340 RETURN

```

Each time a statement is printed, the program branches to the sub-

routine, prints a wait message, counts to 3000 (it takes about five seconds), and returns to the main program. (Notice that there isn't any statement between the FOR and NEXT statements. In this case, we're just letting the system sit there and count as a time delay.)

Up to this point, we've only used END statements occasionally since they don't have to be put at the end of simple programs. However, in this case you *must* put an END statement at the end of the "main" program to prevent the subroutine from being executed after the main program is through.

Subroutines allow you to reuse the same section of a program without having to type in the same lines over and over again. That makes for a smaller program that saves space and runs faster. In addition, using many subroutines lets you write programs that are structured in a modular form that's easy to understand. (It's especially helpful if other programmers will be working on the same program.)

Nested Subroutines

Just like FOR-NEXT statements, you can nest subroutines, calling other subroutines from within subroutines. Let's rewrite the previous example to include a nested subroutine that prints out the message while the system is waiting.

```

10  REM A Time Delay Display
20  PRINT"This is the Apple III"
30  GOSUB 300
40  PRINT"A versatile computer system"
50  GOSUB 300
60  PRINT"Especially with Business BASIC"
70  GOSUB 300
80  PRINT"And the Profile Hard Disk"
90  END
300 REM Time Delay Subroutine
310 PRINT"Wait a moment!"
320 FOR X=1 TO 3000
330   NEXT X
340 GOSUB 500
350 RETURN
500 REM Return message subroutine
510 PRINT"Wait's over!"
520 FOR X=1 TO 300
530   NEXT X
540 RETURN

```

} Main program

} Subroutine

} Nested subroutine

Subroutines can be nested up to 23 deep in Business BASIC. Nesting more than 23 will cause a "STACK OVERFLOW" error.

COMPUTED BRANCHING

Both GOTO and GOSUB have variations that let the program branch to a line number or subroutine, depending on the value of a numeric expression.

ON-GOTO and ON-GOSUB

ON-GOTO and ON-GOSUB are statements that let a program branch to one of several lines depending upon the value of a variable. Once again, let's use a simple example:

```

10 REM An ON-GOTO Example
20 PRINT "Press any key from 1 to 6"
30 GET KEY%
40 ON KEY% GOTO 50,60,70,80,90,100
45 PRINT "You didn't do what we told you":GOTO 20
50 PRINT "You Pressed 1"
60 PRINT "You Pressed 2"
70 PRINT "You Pressed 3"
80 PRINT "You Pressed 4"
90 PRINT "You Pressed 5"
100 PRINT "You Pressed 6"

```

To explain how this works, KEY% must be an integer. If it's 1, the program branches to line 50; if 2, it branches to line 60. If KEY% doesn't equal 1 through 6, the program proceeds directly to the next line. (In this case an incorrect entry will cause the program to loop back to the beginning.)

ON-GOSUB works exactly the same way, except you indicate the subroutines to branch to. Here's an example that will give you a delay that increases with the size of the number you press:

```

10 REM An ON-GOSUB Example
20 PRINT"Press 1-5 for delay"
30 PRINT"The larger the number, the longer the delay"
40 GET DELAY%
50 ON DELAY% GOSUB 100,200,300,400,500
60 PRINT"Wait's Over!"
70 GOTO 20
80 END
100 FOR X=1 TO 100
110 NEXT X
120 RETURN
200 FOR X=1 TO 500
210 NEXT X
220 RETURN

```

```

300   FOR X=1 TO 900
310     NEXT X
320   RETURN
400   FOR X=1 TO 1300
410     NEXT X
420   RETURN
500   FOR X=1 TO 1700
510     NEXT X
520   RETURN

```

Branching from the Keyboard

The ON KBD command is used to tell Business BASIC to execute a statement when *any* key on the keyboard is pressed. When Business BASIC encounters an ON KBD statement, it just keeps on going through the program in a normal manner until a key is pressed. At that point, it executes the statement given with ON KBD. Normally this will be a GOTO.

An ON KBD GOTO is similar to a subroutine in that you have to use a RETURN to allow the program to continue from where it left off. Here's a program that continually prints out a message and tells you when you press a key.

```

10   ON KBD GOTO 100
20   PRINT "This is the Apple III speaking"
30   FOR X=1 TO 1000 } Delay loop
40     NEXT X
50   GOTO 10
100  PRINT "You pressed a key!"
110  RETURN

```

Program branches to 100 when you press a key

(Press RESET to stop the program since it's an infinite loop. You can't use CONTROL C since the ON KBD statement treats it as just another key.)

OFF KBD is used to tell Business BASIC to cancel the last ON KBD statement. It allows a program to react to a keypress only during the section of the program you specify.

FUNCTIONS IN BUSINESS BASIC

A very important element of Business BASIC is its *functions*. Functions are handy for writing Business BASIC programs because they save you time. For instance, instead of writing a program that figures the square root of a number, you need only use the SQR function.

A Business BASIC function takes one or more expressions that are called *arguments* and returns a value. With the exception of string functions, the arguments are enclosed in parentheses.

It's important to remember that a function isn't a statement. You have to use it as part of a program statement.

Here's an example:

```
A=SQR(B)
```

In this case, the SQR function takes the value of the variable B as its argument, returns the square root of B, and assigns it to the variable A.

Here are some examples of the results you'd get using some of Business BASIC's functions:

```
)PRINT SQR(643.2)
)25.3614

)PRINT HEX$(726)
)02D6

)PRINT TEN(02D6)
)726
```

Functions can be substituted for variables or constants just about anywhere in a BASIC program. The only place they can't appear is to the left of an equal sign.

One of the powers of Business BASIC is that it has many special functions that make programming easier. Each of Business BASIC's functions are explained fully in Appendix A.

Numeric Functions

There are 15 functions in Business BASIC that return numeric results. These are shown in Table 6-2. The numeric functions include the common trigonometric functions like SIN and COS as well as some familiar math functions like LOG and absolute value.

String Functions

String functions let you manipulate information contained in strings. Business BASIC's twelve string functions, listed in Table 6-3, will become increasingly useful the more programming you do. You'll find a complete explanation of each function in Appendix A.

Table 6-2. Business BASIC Numeric Functions

Function	Purpose
SIN	Returns the sine of an angle.
COS	Returns the cosine of an angle.
TAN	Returns the tangent of an angle.
ATN	Returns the arctangent of an angle.
INT	Returns the largest whole number less than or equal to the argument value.
RND	Generates a random number.
SGN	Returns -1 if the argument is negative; returns 0 if the argument is 0; returns 1 if the argument is positive.
ABS	Returns the absolute value of an argument.
SQR	Returns the positive square root of the argument.
EXP	Raises e (2.718282) to the power indicated by the argument.
LOG	Returns the natural logarithm of the argument.
CONV&	Returns the long integer value of the argument.
CONV	Returns the real value of the argument.
CONV\$	Returns the string value of the argument.
CONV%	Returns the rounded integer value of the argument.

To give you an idea of how string functions are used in Business BASIC, here are a few examples:

```

)PRINT ASC("F")
)70

)PRINT CHR$(70)
)F

)PRINT LEFT$("APPLE",4)
)APPL

)PRINT RIGHT$("APPLE",4)
)PPL

)PRINT MID$("APPLE",2,3)
)PPL

```

Table 6-3. Business BASIC String Functions

Function	Purpose
LEN	Returns an integer value equal to the length of a string expression.
STR\$	Evaluates an arithmetic expression and returns the value as a string equivalent.
VAL	Evaluates a string expression and returns the real number or integer equivalent.
CHR\$	Evaluates an arithmetic expression and returns a one-character string equal to the ASCII value of the expression.
ASC	Returns the ASCII character code equal to the first character of a string expression.
HEX\$	Returns a four-character string equal to the hexadecimal value of the arithmetic expression.
TEN	Returns the decimal value equal to the last four characters (representing a hexadecimal number) of a string expression.
LEFT\$	Returns a string that's composed of the leftmost characters of a string expression. The length of the returned string is specified by an integer in the argument.
RIGHT\$	Returns a string that's composed of the rightmost characters of a string expression. The length of the returned string is specified by an integer in the argument.
MID\$	Returns a string that's composed of characters in the middle of a string expression. The length of the returned string is specified by two integers in the argument that specify the starting and end position of the mid string.
INSTR	Returns the integer position within a specified string of a substring that's included in the argument.
SUB\$	Lets you replace a part of a string with a specified substring.

Defining Your Own Functions

If a function you want to use from within a Business BASIC program isn't part of the "standard" set, you can define your own using the DEF FN (define function) statement. Here's an example that, once defined, can be called from anywhere else within the program and will take the value of a variable, find its square root, and multiply it by 20:

```
)10 DEF FN Finagle(X)=(SQR(X))*20
)20 INPUT "WHAT NUMBER? ";X
)30 PRINT FN Finagle(X)
)40 GOTO 20
)RUN
WHAT NUMBER? 45.34
134.67
WHAT NUMBER?
.
.
.
```

Line 10 defines the function, named "Finagle." It can be called from anywhere in the program by FN Finagle and used repeatedly.

Although this may not sound like a practical example, you'll find numerous occasions when you want to do the same thing over and over again in a program.

Advanced Programming In Business BASIC **7**

The transfer of data to and from the computer is done with a variety of statements, known collectively as *input/output* statements, or *I/O* statements, for short. Most of the time, you'll be receiving data from the keyboard and sending it out to the video display. But you'll also want to send data to a floppy or hard disk as well as to a printer. In this chapter, we'll look at the many ways I/O is performed in Business BASIC, and we'll see how to use some of Business BASIC's advanced features.

FORMATTING TEXT ON THE SCREEN

We've used Business BASIC's PRINT statement numerous times in the last chapter. The PRINT statement displays text on the screen, and by now you probably take it for granted. However, PRINT is a powerful tool for *formatting* data on the video screen — putting it into a form that is easy to use and interpret.

As you have undoubtedly realized by now, the simplest way to display text on the screen is to follow the reserved word PRINT with the text in quotes. For example:

```
)PRINT "This is the Apple III"  
This is the Apple III
```

To display the contents of a variable, just type the variable name after PRINT:

```
)PRINT Sales  
31243.78
```

(In this case, the variable "Sales" was set to this value previously.)
Let's see what else PRINT can do. A normal PRINT statement ends

with a *line feed* and *carriage return*, meaning that the *next* PRINT statement starts printing at the beginning of the next line. Here's an example:

```

)NEW
)10 REM A Quick Counting Program
)20 FOR X=1 TO 10
)30 PRINT X
)40 NEXT X
)50 END
)RUN
1
2
3
4
5
6
7
8
9
10
)

```

Using Semicolons

You can eliminate the carriage return at the end of a line by typing a semicolon ; at the end of your PRINT statement. For example, in the previous program change line 30 to the following:

```

)30 PRINT X;
)RUN
12345678910

```

The semicolon tells Business BASIC to print the next character at the next available position on the screen. As you can see, there aren't any spaces between the individual pieces of data. If you want spaces, you'll have to add them by using a *literal* (a string in quotation marks):

```

)30 PRINT X" ";
)RUN
1 2 3 4 5 6 7 8 9 10

```

Whenever the printed data extends past the eightieth column of the screen, the display automatically wraps around to the next line. For example, run the following program:

```

)NEW
)10 FOR X=1 TO 240
)20 PRINT "X";
)30 NEXT X
)40 END

```

```

)RUN
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Since there are 240 characters that this program prints out, you can see how the automatic carriage return works.

Using Commas

The video display screen that you use with your Apple III is divided into five 16-character-wide *tab fields*. As you'll see shortly, you can change the preset tab stops. However, when you use a comma after your PRINT statement, each piece of data is printed at sequential tab stops (after the first print position). Try the following:

```

)NEW
)10 FOR X=1 TO 5
)20 PRINT X,
)30 NEXT X
)40 END
)RUN
1           2           3           4           5

```

The tab stops are located at columns 16, 32, 48, 64, and 80. If the data to be printed at the last stop (80) is more than a single character, the entire number or string is printed on the next line.

TAB and SPC

Although you can use semicolons, commas, and spaces (" ") to format text with PRINT statements, Business BASIC offers you a number of other ways to do it.

You can specify exactly where to position the tab stops using the TAB statement. TAB tells Business BASIC where to begin printing the next character. Let's use an example:

```

)NEW
)10 Sales=31243.78
)20 Employees%=45
)30 Name$="John Smith"
)40 PRINT Sales;TAB(11);Employees%;TAB(24);Name$
)50 END
)RUN
31243.78  45           John Smith

```

The number in parentheses specifies the column number to print the *next* item in.

SPC is short for "space" and works in a similar manner. The difference is that it tells Business BASIC how many spaces to leave *between* items.

For example, change line 40 in the previous program to the following and run it:

```
)40 PRINT Sales;SPC(4);Employees%;SPC(8);Name$
)RUN
31243.78   45           John Smith
```

Each SPC statement can specify a maximum of 255 spaces, but you can string several SPC statements together to move the printing position down the screen:

```
)PRINT SPC(255)SPC(255)SPC(100);"Hello"
```

In this example, "Hello" is printed 610 character positions from the beginning.

ADVANCED FORMATTING — PRINT USING _____

Although PRINT, TAB, SPC, and the associated punctuation (, and ;) help you format information, PRINT USING allows you much greater control over the format of data.

At first glance, PRINT USING seems complicated. The easiest way to understand it is to think of creating a *template* (called a *format specification*) that tells Business BASIC how to format groups of text or numbers.

To see the power behind PRINT USING, let's first look at how a group of numbers are printed out using a standard PRINT statement. Type this program:

```
NEW
10 REM Display numbers
20 FOR X=1 TO 5
30 READ Number
40 PRINT Number
50 NEXT X
60 DATA 134.56,45000,3213,-.09,7654.34
70 END
RUN
34.56
5000
213
.09
654.34
```

Although all the numbers are listed, it's difficult to interpret the relationships among them. If you are talking about money, you can add a dollar sign in front of each number by changing line 40 to

```
)40 PRINT "$";Number
)RUN
$134.56
$45000
$3213
$-.09
$7654.34
```

This may help, but what you really want to do is have all the decimal points lined up. That's where the PRINT USING statement comes in. For example, change line 40 to the following:

```
)40 PRINT USING "$5#.2#";Number
)RUN
$ 134.56
$45000.00
$ 3213.00
$   -0.09
$ 7654.34
```

The results are much easier to read. The string in line 40 is the format specification that tells Business BASIC how to print the data. Let's look at it more closely.

The dollar sign tells Business BASIC that it should place a dollar sign before all the numbers; “#” is a place holder, and the number 5 before it reserves five places to the left of the decimal point. (The longest number in the DATA list contains five digits to the left of the decimal point.) The “2#” tells Business BASIC to reserve two digits to the right of the decimal point since we're talking about money. Numbers with more than two digits to the right of the decimal point are automatically rounded off.

You can also type all the individual place holders. In this case, line 40 would become

```
)40 PRINT USING "$#####.##";Number
```

Whether you type out all the place holders or use a number to specify the number of places is purely a matter of choice. Although using numbers is the shortest way (and important when you're trying to save memory space), typing the whole line of place holders lets you visualize how the final printed version will look.

IMAGE

Although you can include your template within a PRINT USING statement, there's a handier way to do it, especially if you'll be using several different formats within the same program. IMAGE statements set up templates that can be referred to later. For example, add a new line to the previous program and change line 40 as follows:

```
)15 IMAGE $5#.2#
)40 PRINT USING 15;Number
```

This is the way your new program will look:

```
10 REM Display numbers
15 IMAGE $5#.2#
20 FOR X=1 TO 5
30 READ Number
40 PRINT USING 15;Number
50 NEXT X
60 DATA 134.56,45000,3213,-.09,7654.34
70 END
```

The PRINT USING statement in line 40 refers back to line 15 for its template. If you want to use several different formats at various places in your program, you can use several IMAGE statements and refer to them by line number.

IMAGE is also useful for cases where you want to use several formats in a single line. Here's an example:

```
)10 IMAGE $6#.2#, 6#.4#
)20 A=23.456
)30 B=8422.32
)40 PRINT USING 10; A,B
)50 END
)RUN
$ 23.46 8422.3200
```

The PRINT USING statement in line 40 must have two real numbers to print since the IMAGE statement in line 10 specified two numbers. Notice that in the case of the first number, the three digits to the right of the decimal point are rounded when the image you specify is shorter than the number.

There are a few rules to remember when using an IMAGE statement:

- In an IMAGE statement, quotation marks are *not* required around the format specification.
- IMAGE must be the *only* statement in a Business BASIC program line.

- IMAGE cannot be used for immediate execution, though PRINT USING *spec* can be.

Formatting Numbers

PRINT USING uses three different characters to reserve spaces for digits:

- # Reserves a space for one digit. (Leading zeroes aren't printed.)
- & Reserves a space for *either* a digit or a comma (for long numbers). At least five spaces to the left of the decimal point must be reserved.
- Z Reserves a space for one digit. (Leading zeroes are printed.)

We've already used the “#” to indicate spaces, but you can also use “&” and “Z.” Let's use a few examples to clear up what can be a very confusing concept.

The only difference between “&” and “#” is that the ampersand (&) causes commas to be inserted into long numbers. For example, try the following:

```

)10 REM Display numbers
)20 FOR X=1 TO 5
)30 READ Number
)40 PRINT USING "$5&.2&";Number
)50 NEXT X
)60 DATA 134.56,45000,3213,-.09,7654.34
)70 END
)RUN
$ 134.56
$ !!!!!!!!!!!
$ 3,213.00
$ -0.09
$ 7,654.34

```

If you don't leave enough spaces for the numbers in a PRINT USING statement, you'll get what amounts to an error message—a line of !!!!!.

Let's look at what happened. The second piece of data in the DATA statement is 45000. Although that's a five-digit number, it becomes six characters long when you add the comma. Since the format specification only allowed for five characters, an error resulted.

Our point here is that while you're designing displays using PRINT USING, you'll have to make absolutely sure that you allow for the longest number in the program.

We can solve the previous problem by changing line 40 to

```

)40 PRINT USING "$6&.2&";Number
)RUN
$ 134.56
$45,000.00
$ 3,213.00
    -0.09
$ 7,654.34

```

When used in a format specification, a "Z" is similar to a "#" except that a "Z" causes a number to be printed with leading zeros. To see this, change line 40 in the previous program to

```

)40 PRINT USING "$6Z.2Z";Number
)RUN
$000134.56
$045000.00
$003213.00
$0000-0.09
$007654.34

```

So far, we've covered how PRINT USING is used with fixed-point numbers. In the Apple III, a fixed-point number is *any* number that isn't displayed with an exponent. Integers, long integers, and real numbers are all fixed-point numbers.

In addition to the characters that specify the digits "#," "&," and "Z," there are also seven other characters used with fixed-point numbers in PRINT USING statements:

- + Reserves a character position for the sign.
- Reserves a position for the minus sign. (Only used if the number is negative.)
- \$ Reserves a position for the dollar sign.
- ** Replaces leading space with asterisks.
- ++ Reserves the rightmost print positions for signs and dollar signs (if there are any).
- Minus sign printed only if the number is negative.
- \$\$ Reserves the rightmost print position for a dollar sign and +/- sign (if there are any).

Let's look at how a few of these characters are used.

A "+" causes a plus sign to be printed in front of all positive numbers and a minus sign to be printed in front of all negative numbers. To see this, change line 40 in the previous example to the following:

```

)40 PRINT USING "$+5#.2#";Number
)RUN
$+ 134.56

```

```

$+45000.00
$+ 3213.00
$-   0.09
$+ 7654.34

```

A “-” causes a minus sign to be printed in front of all negative numbers. For instance, change line 40 to

```

)40 PRINT USING "$-5#.2#";Number
)RUN
$  134.56
$ 45000.00
$  3213.00
$-   0.09
$  7654.34

```

Using “-” in the format specification causes the minus sign to be printed off to the far left, instead of directly in front of the number.

Using “**” in PRINT USING causes asterisks to be printed instead of leading spaces. To see this, change line 40 to

```

)40 PRINT USING "**5#.2#";Number
)RUN
**134.56
45000.00
*3213.00
***-0.09
*7654.34

```

Scientific Notation

Scientific notation is a shorthand for specifying very large numbers. (See Chapter 6 for a more complete explanation.) You use scientific notation in a PRINT USING specification by using “E” and specifying either a single digit or none to the left of the decimal point. In addition, there must be three or four digits allowed for the exponent. Here are two examples:

```

)PRINT USING "#.3#3E";231.23423
2.312E+2
)PRINT USING ".3#3E";231.23423
.231E+3

```

The number in front of the “E” in each of the examples must be included to specify the number of digits in the exponent. Note, however, that two character positions are always occupied by the “E” and a sign. Therefore, you must always specify three or four digits in the exponent.

Engineering Notation

Engineering notation is similar to scientific notation, except that the exponent is forced to be a *multiple of 3*. You specify engineering notation by using "E" and specifying three digits to the left of the decimal point. Digit positions are only important to the left of the decimal point. You can use either a "#" or a "Z" to indicate digit positions. In this case, the "Z" causes leading zeroes to be printed.

For example:

```
)PRINT USING "3#.4Z4E";1234.34
1.2343E+03
)PRINT USING "3Z.3#4E";.02314
023.140E-03
```

SCALE

The SCALE function is used with PRINT USING to shift the decimal point in a number left or right. It uses two arguments in parentheses. SCALE takes ten raised to the power of the first argument and multiplies it by the second argument. Here's an example:

```
)PRINT USING "8#.3#";23145.67
23145.700
)PRINT USING "8#.3#";SCALE(3,23145.67)
23145700.000
)PRINT USING "#8#.2#";SCALE(-3,23145.67)
23.15
```

You can see that SCALE essentially moves the decimal point left or right depending on its first argument.

When would you use SCALE? As we mentioned earlier, Business BASIC works much faster with integers than with real numbers. To take advantage of this fact, you could convert dollars and cents into cents to do a numeric calculation and then use SCALE in a PRINT USING statement to print out the results with the decimal point positioned correctly.

PRINT USING with Strings

Although PRINT USING is handiest when you're printing out numbers, a few of its special features are useful for formatting text on the screen. There are three PRINT USING elements that work with strings:

- A Left-justify
- R Right-justify
- C Center

You use PRINT USING with strings by specifying the length of the *field* followed by “A,” “R,” or “C” to specify whether the text will be left-justified, right-justified, or centered within the field. For example, suppose you want to center a string in the middle of the screen (80 characters):

```
)PRINT USING "80C";"The Apple III"
           The Apple III
```

If you use several fields in an IMAGE statement, each subsequent field starts where the preceding one left off. To see this, try the following:

```
)10 IMAGE 5A, 5C, 5R
)20 PRINT USING 10;"The","Apple","III"
The Apple III
```

“5A” prints “The” left-justified in a five-character field; “5C” prints “Apple” centered in a five-character field; and “5R” prints “III” right-justified in a five-character field.

Thus, you can see that PRINT USING is by far the most powerful way of formatting text. The best way to find out about it is to experiment.

CHR\$ — PRINTING CHARACTERS IN ASCII

CHR\$ allows you to generate both visible and invisible characters from within a Business BASIC program. The “beep” is an example of an invisible (or non-printing) character, and a double quote mark is an example of a visible character. You can do this because the Apple III stores characters as a series of 1s and 0s. The standard code for storing characters is called ASCII, which is short for American Standard Code for Information Interchange. CHR\$ lets you specify the ASCII code number for a particular character.

Because the Apple III has nearly all of the visible characters of the ASCII character set on the keyboard, you won’t need to use CHR\$ too often for generating characters. One exception is the double quote. You can’t include a double quote within a text string because Business BASIC will interpret the second double quote as the end of a string. By

using `CHR$(34)`—the ASCII code for a double quote is 34—you can print it. Here are two examples of the same string, one with and one without quotes:

```
)PRINT "The Rain in Spain"
The Rain in Spain
)PRINT CHR$(34);"The Rain in Spain";CHR$(34)
"The Rain in Spain"
```

You can also use `CHR$` to send a “beep” to the built-in speaker. You might do this to call attention to something in your program. The ASCII code for the beep is 7, so the statement you would use is

```
)PRINT CHR$(7)
```

You'll find a complete list of the ASCII codes in Appendix J. Codes 0 through 31 are invisible codes that don't appear on the screen. Many of them are used to control the console. Codes 32 to 127 are visible characters, nearly all of which can be generated from the keyboard.

DISPLAYING TEXT

Although the numerous variations of `PRINT` and `PRINT USING` are the most versatile way of formatting characters on the screen, there are a number of additional Business BASIC statements that control how characters appear on the screen.

HOME

You may have noticed that your screen display can become crowded after you've been working for a while. For example, it can get messy after a long listing.

To clear it, just type **HOME**. This will clear the screen and position the cursor in the upper left-hand corner of the screen.

INDENT

`INDENT` is a *reserved variable* that controls how program statements and `FOR-NEXT` loops are indented from the line numbers. As we saw in the last chapter, indentation allows a more readable program by setting off `FOR-NEXT` loops so that they can easily be spotted in a long program.

Let's review. Here's a short program using a FOR-NEXT loop:

```

)NEW
)5 REM This is a short program
)10 FOR X=1 TO 100
)20 PRINT X/2
)30 NEXT X
)40 END

```

But something different happens when you print it out. Type **LIST** and you'll see

```

5  REM This is a short program
10  FOR X=1 TO 100
20  PRINT X/2
30  NEXT X
40  END

```

Business BASIC automatically does the indenting (the default value is 2), but you can change the indentation with an **INDENT** command. For instance:

```

)INDENT=5
)LIST

5      REM This is a short program
10     FOR X=1 TO 100
20     PRINT X/2
30     NEXT X
40     END

```

Changing the reserved variable **INDENT** results in more indentation. Notice that in addition to the indented FOR-NEXT loop, the beginning of the program lines are also indented farther from the line numbers. (You could use an **INDENT** command within a program, but it makes more sense to use it in the immediate mode.)

INDENT stays at the value you've set until you assign it a new value or turn the Apple III off. When you boot up the system, **INDENT** returns to its default value of 2.

OUTREC

OUTREC lets you control the number of spaces that will print out on one line on a printer. For instance, many printers do not have automatic wraparound but just run off the end of the paper, dropping the end of the line. Let's use an example. If you type **OUTREC=40**, the printer will do a line feed and carriage return after every 40 characters.

WINDOW

If you think of your video display as a “window” into the computer, you’ll quickly understand what the WINDOW command does. The normal window fills the entire screen, but the WINDOW command lets you specify a much smaller window or one that’s just about any size you might want.

The normal window is 80 columns wide by 24 lines deep. To make it smaller, you could type

WINDOW 20,5 TO 60,18

To help you understand what’s going on here,

- 20 is the starting column number of the new window
- 5 is the starting row number of the new window
- 60 is the ending column number of the new window
- 18 is the ending row number of the new window.

The new window would appear as shown in Figure 7-1.

If you type CAT you’ll see the listing appear only in the middle of the

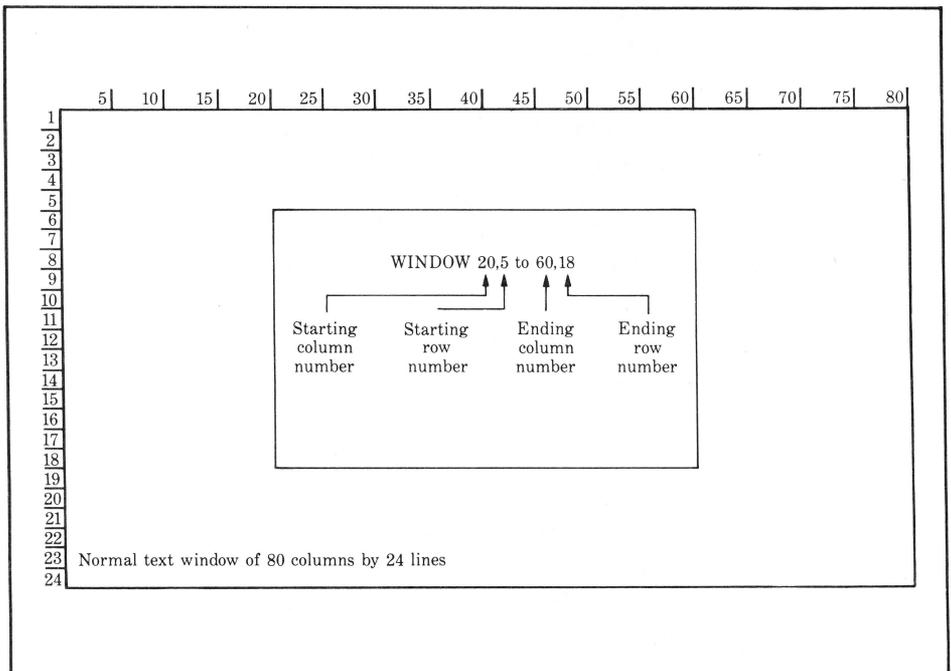


Figure 7-1. Changing the display window

screen. WINDOW is handy for customizing the look of a program, although you'll have to do some planning with graph paper to get the window to appear *exactly* where you want it to. For instance, you might want the first messages of a program to appear in the middle of the screen. If you want to change things back to normal, just type TEXT.

TEXT

TEXT restores the size of the video display window to normal (80 columns by 24 lines). If you used the WINDOW command to change the screen, TEXT will return it to normal. TEXT will also cancel the graphics modes (a subject covered in Chapter 8).

VPOS and HPOS

Although WINDOW changes the display area on the video display, you can also change the position of the cursor *within* that window by using VPOS and HPOS.

VPOS stands for vertical position, and HPOS stands for horizontal position. Although it's possible to use values of 0 to 255 for each of these, numbers higher than 80 (for HPOS) and 24 (for VPOS) don't make much sense since the display is 80 characters wide by 24 lines deep.

As an example, VPOS=8 and HPOS=20 would move the cursor down to the eighth line and twentieth column on the display, as shown in Figure 7-2.

Since both statements are reserved variables, you can also find out where the cursor is currently positioned by typing PRINT HPOS or PRINT VPOS.

INVERSE and NORMAL

There will be times when you want to highlight a section of text. A handy way to do this is with the INVERSE command. When used within a program, it makes all subsequent characters appear as black characters on a white (or green) background.

The NORMAL command returns things to "normal." To see how it works, type in the following program and run it:

```

)10 REM INVERSE VS. NORMAL TEXT
)20 INVERSE
)30 PRINT "THIS IS INVERSE TEXT";
)40 NORMAL: PRINT
)50 PRINT "NOW IT'S BACK TO NORMAL"
)60 END

```

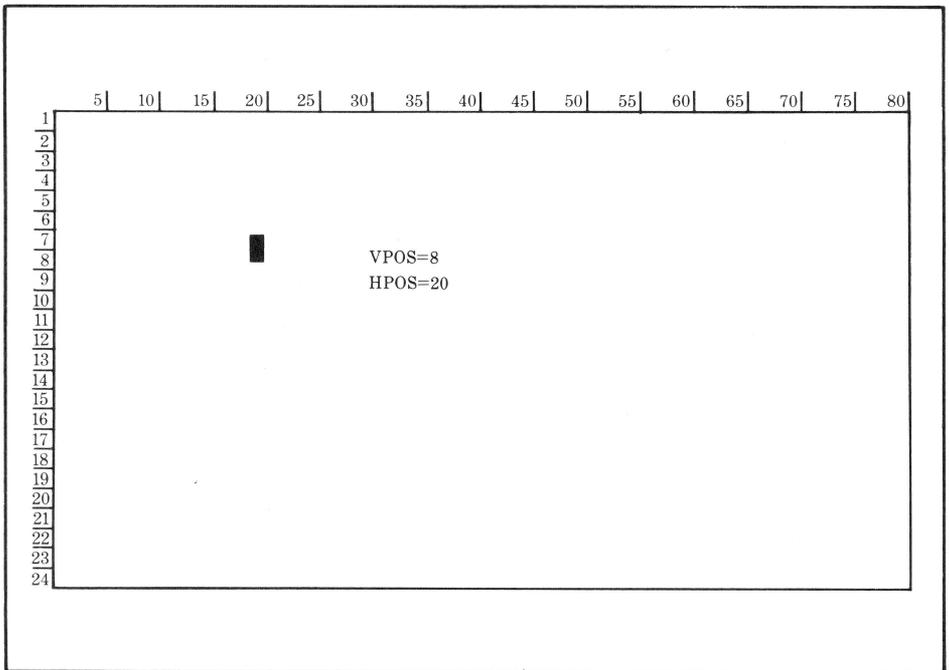


Figure 7-2. Positioning the cursor

Note that in this example, the extra PRINT statement in line 40 acts like a RETURN and leaves a space before the next line of text.

WORKING WITH FILES IN BUSINESS BASIC

There are a number of different types of files that you will be working with in Business BASIC. Recall from Chapter 3 that a file is an organized collection of information. This information may reside on a device like a floppy disk drive or a hard disk drive. In addition, the Apple III treats each device connected to the computer as a file. File devices include the keyboard and screen (the console), floppy disk drives, the Profile disk drive, and so forth.

While the programs that you write can be saved in files called program files, there are several kinds of files that can be created from within a Business BASIC program. These are used to store the data (text and numbers) that your program operates on.

Naming Files

Let's review a bit. In the Apple III, a file is referred to by its *filename* or *pathname*. As we explained in Chapter 3, a filename begins with a letter and can be as many as 15 characters long. The filename of a device begins with a period. Some examples are .PROFILE, .D1, and .CONSOLE.

A pathname is the "path" the computer takes to find the file. A pathname begins with a slash (/) and a volume or device name, followed by the names of any subdirectories leading to a particular file.

PREFIX\$

As you've found out by now, a full pathname can be quite long. Referring to a file by typing something like the following every time is time-consuming:

```
.PROFILE/Inventory/Appliances/Refrigerators
```

But there's a shorthand way to specify a pathname: you can assign the pathname to the reserved variable PREFIX\$. For instance, typing

```
)PREFIX$=.PROFILE/Inventory/Appliances/Refrigerators
```

lets you refer to the pathname by using PREFIX\$ from anywhere within the program. When you boot your Business BASIC disk, PREFIX\$ is automatically set to the name of the volume in the boot device (for instance, /BASIC). You can find out what that is by typing

```
)PRINT PREFIX$
```

"/BASIC" is the default value assigned to PREFIX\$ by the Business BASIC disk. If you enter a pathname that *does not* begin with a "." or a "/", Business BASIC automatically "assumes" that you want to use the prefix that's stored in PREFIX\$. For instance, if you have a file named "CHECK.BALANCE" all you need to do to run it is type

```
)RUN CHECK.BALANCE
```

Business BASIC uses the pathname stored in PREFIX\$ as a prefix to any filename you specify. Since "/BASIC" is the default PREFIX\$, RUN CHECK.BALANCE is equivalent to

```
)RUN /BASIC/CHECK.BALANCE
```

You have to specify the complete pathname if you don't want to use the pathname stored in PREFIX\$.

CREATE

CREATE is used to create files. You specify the file you want to create using the pathname and the *type* of file you want to create. You can create three types of files in Business BASIC:

- Text
- Data
- Catalog.

In Business BASIC programs, the most common types of files you'll be using will be text and data files.

Text files are comprised of text—strings of ASCII characters. Numbers stored in a text file are converted to strings. Data files usually contain numeric information that can be assigned to a numeric variable within a Business BASIC program; however, data files can also contain strings. Text files are used when you are working with lots of strings; data files are more appropriate when you are working with numbers. A catalog file contains a directory or subdirectory of files.

Text and data files are used to store the data that a program uses. All of the programs that we've worked with so far either have had the data contained in a DATA statement or have gotten information from the keyboard using an INPUT statement. However, when you are working with large amounts of data, it is easier to store the data in a file separate from the main program.

For example, if you have a program to balance your checkbook each month, it would be impractical to have to keep changing DATA statements each time you want to record a check. By keeping the information for each month's checks in an individual data file, you can avoid having to change the main program at all. Separate data files also make the data accessible to a number of different programs. For instance, you could have separate programs to balance the checks, prepare monthly statements, and so on. To create a text file on your BASIC disk you'd use a command like this one:

```
)CREATE "/BASIC/Instruct/", TEXT
```

The quotation marks are *required* when you use a CREATE statement in a program, but they aren't required for immediate execution.

Don't forget that you can leave off the pathname that's stored in PREFIX\$. Therefore, if you haven't changed PREFIX\$ after booting the system, the following CREATE statement will do exactly the same thing:

```
)CREATE "Instruct", TEXT
```

You can also create a catalog file for a new subdirectory named "Games" by typing

```
)CREATE "Games", CATALOG
```

or

```
)CREATE "/BASIC/Games/", CATALOG
```

Specifying the Size

The data in a file is organized into units called *records*. The size of a record in Business BASIC can be anywhere from 3 to 32767 bytes. With one exception (see "Random Access Files" later in this chapter) you won't have to worry about the size of each individual record. If you don't specify the size of a record when you create a file, it's automatically set to 512 bytes. Here's an example of how you specify the size:

```
)CREATE "Instruct", TEXT, 1028
```

This creates a file named "Instruct" with records that are each 1028 bytes long. A record size is only *required* for a random-access file (which we'll talk about later in this chapter).

If you create a catalog file, the default record size is also 512 bytes—enough for 12 files in your subdirectory. If you think you'll have more than 12 files in the subdirectory, specify a larger size. (1028 will allow you to include about two dozen files.)

MANAGING FILES

There are a number of commands in Business BASIC that let you organize and keep track of the files you've created. The job of managing the files on your disks is often referred to as *housekeeping*.

CATALOG

When you boot the Business BASIC disk, you see a listing on your video display of all the files contained on your Business BASIC disk. This is the equivalent of typing **CAT** (or **CATALOG**) in immediate mode.

Just typing **CAT** displays the root directory of the specified disk (see

Chapter 3). If you want to see the contents of a subdirectory, you specify the pathname. For example,

```
 )CAT /BASIC/Games
```

or

```
 )CAT Games
```

will show you the list of files in the "Games" subdirectory.

Types of Files

When you boot the Business BASIC disk, you'll see quite a few files of different types. We talked about the three (text, data, and catalog) that you can create in Business BASIC. But Table 7-1 contains a complete list of the other types you might run into.

Deleting Files

It's good programming practice to do occasional housekeeping and remove files or subdirectories that you're not using anymore. This is

Table 7-1. Types of Disk Files

File type	Contents
BAD	A damaged area of the disk that's been locked out by the operating system.
BASIC	A BASIC program created with the SAVE statement.
BINARY	An assembly language file.
CAT	A root directory or subdirectory.
DATA	A Business BASIC data file.
FONT	A file that contains type font information.
FOTO	A graphics "picture".
PASCOD	A Pascal program file.
PASDTA	A Pascal data file.
PASTXT	A Pascal text file.
RESERV	Reserved for future use.
SYSTEM	An SOS program file.
TEXT	A Business BASIC text file.
UNKNWN	A Business BASIC data or text file that was opened but not written to.

done using the DELETE command. Storing files that you don't need wastes valuable storage space on your disks.

You can delete individual files or entire subdirectories; however, it's important to remember that you must delete all the individual files within a subdirectory before you delete the subdirectory.

Here are some examples of how DELETE is used:

```
)DELETE /BASIC/Instruct
```

or

```
)DELETE Instruct
```

deletes the file named "Instruct".

If you have a subdirectory named "Games" and you've deleted all the individual files within the subdirectory, you can enter

```
)DELETE /BASIC/Games
```

or

```
)DELETE Games
```

to delete the entire subdirectory.

Changing File Names

You can use RENAME to change the names of files, volumes, and subdirectories. The form of the command is

```
RENAME old name, new name
```

Suppose you wanted to change the name of a file named "Instruct" to "Documents". You would use the command

```
)RENAME Instruct,Documents
```

Protecting Files — Lock and Unlock

It's relatively easy to accidentally erase an important file. Once it's gone, there's no way you can get it back unless you've made a backup disk. That's why it's important to protect files. You do this using the LOCK command. Once a file is locked, you can't delete, rename, write to, or save to that file name until you unlock it with the UNLOCK command. Here's an example:

```
)LOCK /BASIC/Instruct
```

or

```
)LOCK Instruct
```

locks the file, and

```
)UNLOCK /BASIC/Instruct
```

or

```
)UNLOCK Instruct
```

unlocks it. A locked file is listed in the catalog with an asterisk (*) in front of the file type.

Remember you can lock the entire contents of a floppy disk by placing a tab over the write-protect notch on the disk.

USING FILES IN PROGRAMS

To help you understand how to use a file in a program, you can compare it to using the common file folder found in every office filing cabinet. Before you use the information in the folder, you have to *open* the file; and before you can put the folder away, you have to *close* it. Much the same thing must be done with Business BASIC files.

OPEN#

You open a Business BASIC file by using the OPEN# statement. In Business BASIC, you can have as many as ten files open at the same time. You follow OPEN# with a number from 1 to 10 (which assigns the file a number you can refer to from that point on) and the pathname. Here are a few examples:

```
110 OPEN#1, .Console *
120 OPEN#2, .Printer
130 OPEN#3, Instruct
140 OPEN#4, Widgetcount
```

OPEN# Options

You can also specify *how* the file you've opened will be used. There are three options:

AS INPUT The file can only be read from.

AS OUTPUT The file can only be written to.

AS EXTENSION Lets WRITE# or PRINT# add information to the end of the existing file.

Here are some examples:

```
100 OPEN#1 AS INPUT, Instruct
110 OPEN#2 AS OUTPUT, .Printer
120 OPEN#3 AS EXTENSION, Videogame
```

If you don't specify an option, the file you open can be written to and read from as long as the type of file supports reading and writing. For instance, you can only send data *to* a printer.

CLOSE and CLOSE#

You *must* close all open files with the CLOSE or CLOSE# statement before ending the program. If you don't close the files, you lose the information you put into it (like an open office file results in papers scattered on the floor). Closing a file can be done in one of two ways:

CLOSE Closes *all* open files.

CLOSE# Closes the specified file number.

SEQUENTIAL- AND RANDOM-ACCESS FILES _____

There are two different types of data files: *sequential-* and *random-access*. What differentiates the two is the method in which the files are stored. Sequential-access files allow you to read and save information sequentially, while random-access files allow you to select files at random to read or write. In the following sections, we'll cover both types in detail.

Sequential-Access Files

Recall that information in a file is organized into units called records. In a sequential file, these records are stored sequentially. In order to locate a record in a sequential file, you must search through the file from the beginning until you find the record that you want. A good example of sequential file storage is a cassette music tape. In order to find the song you want to hear, you must search through the tape from the beginning.

Sequential-access files are the most efficient way to store data, since regardless of how much data is stored in each record, there is no “wasted” space on the disk.

There are, however, a few major disadvantages. A sequential access can take a long time. If you only have a small file with a few records, that's not a problem. But if you have disk full of data, it can take a *long* time to find a specific record. In addition, you can only *add* data to the end of a file; and it's often impossible to *change* individual records. For this reason, sequential files are best for storing data that you won't want to change in the future.

There are a few rules you should remember when you're working with sequential files:

- When you first open a sequential file, data is read from or written to the beginning of the file. Each subsequent time you read from or write to the file, the access begins where the last one left off.
- When you open a file with the option AS EXTENSION, any new information is written at the *end* of the file. Each additional access starts where the previous one left off.

Random-Access Files

A *random-access* file is made up of records that are each exactly the *same size*. In a random data file, each record is broken up into units called *fields*, with each field containing a number or a string. Random text files, on the other hand, simply consist of a series of bytes with each byte representing a single string character.

To visualize a random-access file, you can think of a floppy disk divided into sections that are all the same length. Since each record takes the same amount of space (whether or not it's filled with data), you can easily select a record by number without having to search through any others.

You use CREATE in exactly the same way whether you're creating a sequential- or random-access file. However, if you don't specify the size of a record when you create a file you'll be using for random-access, Business BASIC will automatically assign each record a length of 512 bytes.

If you're concerned about saving memory space, planning the record size is important. Using data that's much smaller than the record size wastes memory space. The size of the record has to be large enough to

contain all the data that you'll be writing into the record. Here's the number of bytes that each type of data in a random-data file requires:

Integer	3 bytes
Real number	5 bytes
Long integer	9 bytes
String	Length of string plus two bytes.

For instance, if you're storing check information in a record, you must figure the *maximum* size of each individual entry (field) and add them up to get the total number of bytes per record. For example:

Check Number:	0000 (3 bytes)
Amount:	00000.00 (5 bytes)
Date:	MM/DD/YY (10 bytes)
Payee:	Longest entry (25 bytes)

In this case, the amount of memory required for each record of the file (43 bytes) is *much* smaller than the default value of 512 bytes. To save memory space you could specify a record size of 43 bytes in the CREATE statement used to create the file.

If you'll be accessing a random text file, it's a good idea to plan the size of a record so that it can contain the longest line of text you'll be saving.

Here are several important points you should remember about random-access files:

- When you access a random record, reading or writing begins at the first field of the record.
- If a field won't fit into the space remaining in a record of a data file, it's written into the next record. However, if a field is too big to fit in a record, you'll get an "?OUT OF DATA" error.

GETTING INFORMATION INTO FILES

Business BASIC uses three statements to write information *into* a file:

PRINT# writes information to a text file.

PRINT# USING writes formatted information to a text file in the same way as PRINT USING writes to the screen.

WRITE# writes information to a data file.

PRINT#

PRINT# allows you to send ASCII data to a text file, printer, or other file device. One line of text is sent to the file for each expression you specify. The following program creates a sequential-access file and writes information to it:

```
10 REM Sequential Text File Example
20 CREATE "T.FILE", TEXT
30 OPEN#1 AS OUTPUT, "T.FILE"
40 FOR RECORD=1 TO 20
50 PRINT#1; "THIS IS NUMBER ";RECORD
60 NEXT RECORD
70 CLOSE #1
80 END
```

To create a random-access file with PRINT#, you need to specify the record number in the PRINT# statement. For instance, changing line 50 to the following:

```
50 PRINT#1,RECORD; "THIS IS NUMBER ";RECORD
```

will create a program that begins writing the data in record 1 and ends with record 20. Anytime you specify a *record number* in a PRINT# statement, Business BASIC will automatically create a random-access file.

NOTE: In this example you would also want to specify a much smaller size for each record in order to save space. Since you need 17 bytes for the string (string length plus 2) and 3 bytes for the integer, changing line 20 to

```
20 CREATE "T.FILE", TEXT, 20
```

will save a great deal of space.

PRINT# USING

PRINT# USING writes formatted data to text files in the same way PRINT USING writes to the screen. The same characters used to create the format specification in PRINT USING statements will work with the PRINT USING# statement. Refer to the discussion of PRINT USING earlier in this chapter for a list of these characters.

WRITE#

WRITE# writes information into a *data file* as opposed to PRINT#, which writes information into a text file. Let's modify the previous example to create a sequential data file:

```
10 REM Sequential Data File Example
20 CREATE "D.FILE", DATA
30 OPEN#1 AS OUPUT, "D.FILE"
40 FOR RECORD=1 TO 20
50 WRITE#1; "THIS IS NUMBER ",RECORD
60 NEXT RECORD
70 CLOSE#1
80 END
```

Note that the CREATE statement in line 20 creates the DATA file and the WRITE# statement in line 50 writes information into the file. To create a random-access data file, you would specify a record number in the WRITE# statement.

When you use the WRITE# statement to write information to a file in a random-access manner, the new information *replaces* the previous information in the record.

GETTING INFORMATION FROM FILES

Once you've entered data into a file, you obviously need a way to get it out. Business BASIC uses the following statements to read data from a file:

INPUT# reads information from a *text* file.

READ# reads information from a *data* file.

INPUT#

You use INPUT# to read information from a text file. For each variable, INPUT# reads a single line of ASCII text. Let's write a program to retrieve the text we stored earlier with PRINT#. To review, here's what the original program looked like:

```
10 REM Sequential Text File Access
20 CREATE "T.FILE", TEXT
30 OPEN#1 AS OUTPUT, "T.FILE"
40 FOR RECORD=1 TO 20
```

```

50 PRINT#1; "THIS IS NUMBER ";RECORD
60 NEXT RECORD
70 CLOSE#1
80 END

```

Assuming you entered and ran this program, you'll have a sequential text file called "T.FILE" on your disk. Here's a program that will read the text from the file, into the Apple III's memory and display it on the video screen:

```

10 REM Read T.FILE
20 OPEN#1 AS INPUT, "T.FILE"
30 ON EOF#1 GOTO 70
40 INPUT#1; ERIN$
50 PRINT ERIN$
60 GOTO 40
70 CLOSE#1
80 END

```

```

)RUN
THIS IS NUMBER 1
THIS IS NUMBER 2
THIS IS NUMBER 3
THIS IS NUMBER 4
THIS IS NUMBER 5
THIS IS NUMBER 6
THIS IS NUMBER 7
THIS IS NUMBER 8
THIS IS NUMBER 9
THIS IS NUMBER 10
THIS IS NUMBER 11
THIS IS NUMBER 12
THIS IS NUMBER 13
THIS IS NUMBER 14
THIS IS NUMBER 15
THIS IS NUMBER 16
THIS IS NUMBER 17
THIS IS NUMBER 18
THIS IS NUMBER 19
THIS IS NUMBER 20

```

This program sets up a loop to read a record and print it out until the end of the file (EOF) is reached.

If you created a random-access text file, you'll need a slightly different program to retrieve the data from the file. Just as specifying a record number for PRINT# tells the program where to start writing data, including the record number in an INPUT# statement tells Business BASIC where to start reading data. You must specify the number(s) of the record(s) you want to read. Assuming you created a random-access file that started writing text data at record 1, the following program will retrieve the data:

```

10 REM Retrieve Random Text
20 OPEN#1 AS INPUT, "T.FILE"
30 FOR COUNT=1 TO 20
40 INPUT#1,COUNT; MESSAGE$
50 PRINT MESSAGE$
60 NEXT COUNT
70 CLOSE#1
80 END

```

Note that you must specify the number of *every* record you want to retrieve. The most logical way to do this is to use a FOR-NEXT loop with the index variable as the number of the record you want.

READ#

READ# works like INPUT# except that it retrieves information from a *data* file. Although reading a data file is similar to reading a text file, there are a few subtle differences. Let's review the program we used to generate a sequential-access data file.

```

10 REM Sequential Data Access
20 CREATE "D.FILE", DATA
30 OPEN#1 AS OUTPUT, "D.FILE"
40 FOR RECORD=1 TO 20
50 WRITE#1; "THIS IS NUMBER ",RECORD
60 NEXT RECORD
70 CLOSE#1
80 END

```

It's important to remember that unlike the program that created the text file, this program creates a file where the *data* is stored as two separate fields—a string and an integer. Therefore, you'll need a program that retrieves and prints out *both* fields.

```

10 REM Retrieve Sequential File
20 OPEN#1 AS INPUT, "D.FILE"
30 FOR COUNT=1 TO 20
40 READ#1; MESSAGE$,NMBR%
50 PRINT MESSAGE$,NMBR%
60 NEXT COUNT
70 CLOSE#1
80 END

```

This will print out the entire contents of the file we created.

If you created a random-access data file, you'll have to include a record number in the READ# statement in order to read the data from the file. One way to read the data is with the following program:

```

10 REM Retrieve Random Access Data File
20 OPEN#1 AS INPUT, "D.FILE"

```

```

30 FOR RECORD=1 TO 20
40 READ#1,RECORD; MESSAGE$,NMBR%
50 PRINT MESSAGE$,NMBR%
60 NEXT RECORD
70 CLOSE#1
80 END

```

Directing Output — OUTPUT#

In Business BASIC, all output is normally sent to the video display. However, there might be times when you want to keep a record of the screen output on a disk file or send it to a printer. You can do that using the OUTPUT# statement.

Suppose you want the printer to print everything that would normally be displayed on the screen. You can use the following statements:

```

10 OPEN#1 AS OUTPUT, .Printer
20 OUTPUT#1

```

It's not really necessary to use the option "AS OUTPUT" since the printer is an output device only. However, it's good programming practice to specify the type of file when you open one.

DETECTING THE END OF A FILE _____

There are times when it is useful to detect the end of a file, particularly when you are reading a sequential file. In Business BASIC, the ON EOF# and OFF EOF# statements allow you to control the detection of the end of file (EOF) condition.

ON EOF#

This statement tells Business BASIC what to do if it reads past the end of a file. If you haven't used the ON EOF# statement, you'll get an "?OUT OF DATA" error if Business BASIC tries to read past the end of a file. Most of the time, you'll want the program to branch to a location or print out a more detailed message on the screen, such as

```
220 ON EOF#2 GOTO 100
```

or

```
310 ON EOF#4 PRINT "No More Data - Aborting"
```

Note that you *cannot* use a RESUME statement with ON EOF# like you can with ON ERR.

OFF EOF#

OFF EOF# cancels the last ON EOF# statement that was executed. If an end of file is encountered by Business BASIC after the OFF EOF#, the error message is displayed and the program stops. To cancel the ON EOF examples in the previous section, you'd use

```
250 OFF EOF#2
```

```
350 OFF EOF#4
```

RUNNING A PROGRAM AUTOMATICALLY

There are several ways to make a program execute automatically. You can use a CHAIN statement in one BASIC program to execute a second BASIC program. You can also execute a series of RUN commands stored in a text file using the EXEC command.

CHAIN

Even though the Apple III has a tremendous amount of space available for Business BASIC programs, you eventually might want to run a program that is larger than the memory you have.

To do this, you can break the program into smaller sections and run each section using the CHAIN statement.

CHAIN is used in one program to load and run a second program. CHAIN automatically loads and runs the program you specify *without changing* the values of variables or closing any files. You can specify the program to be run by typing CHAIN and the pathname. For instance:

```
2310 CHAIN .D1/CheckBa12
```

You can also specify where the second program should start running by typing a comma and a line number after the pathname:

```
2310 CHAIN .D1/Checkba12,300
```

EXEC

EXEC (for execute) allows you to run a series of programs or execute a number of commands automatically. Normally, the only way you could execute a series of commands would be to enter one command, wait for it to finish, enter another command, and so on.

For example, you could use EXEC to run a series of different programs and then produce a catalog listing. The equivalent commands in immediate mode might be

```
RUN FirstProgram
RUN SecondProgram
RUN ThirdProgram
CAT
```

To use EXEC you need to create a text file containing all the commands you want your Apple III to perform. Once you've created the text file, you type EXEC followed by the name of the text file you've created.

EXEC tells Business BASIC to execute the commands stored in the text file. From that point on, Business BASIC ignores anything you type in from the keyboard with the exception of CONTROL C, which stops the process. CONTROL C, a STOP statement, an end of a file, or an error all cause control to return to the keyboard.

Creating an EXEC File

In order to create a "command file" to use with EXEC, you have to write a short program that creates a text file. Here's an example:

```
10 REM Make up text file for Exec
20 OPEN#1 AS OUTPUT,"TEXT.FILE"
30 INPUT"Enter Text String: ";TXT#
40 IF LEN(TXT#)=0 THEN 60
50 PRINT#1;TXT#:GOTO 30
60 PRINT"Closing Text File":CLOSE
99 END
```

Line 20 opens a file named "TEXT.FILE" (you can name the file anything you want by changing the name in quotation marks) that takes the text strings entered in line 30. Line 40 shows how you can use the LEN function to wait for a RETURN without data. When the program gets a RETURN without data, it closes the file and ends the program.

We suggest that you carefully type in this program and save it on your disk by typing

```
)SAVE TEXT.FILE.MAKER
```

(or whatever name you want to call it). When you run this program, it will create a file named "TEXT.FILE" that "collects" the command lines you entered.

You can enter any valid Business BASIC statement in the text file. A sample run might be something like the following:

```
)RUN TEXT.FILE.MAKER
Enter Text String: RUN FirstProgram
Enter Text String: RUN SecondProgram
Enter Text String: RUN ThirdProgram
Enter Text String: CAT
Enter Text String: {RETURN pressed here}
Closing Text File
)
```

To use the file that you've created, you would enter

```
)EXEC TEXT.FILE
```

At this point, the three programs you specified would run in sequence followed by a catalog listing.

You can also use conditional statements to make one program dependent on the result of another program. For instance, you could put the following in "TEXT.FILE":

```
RUN FirstProgram
IF Value%=1 THEN RUN SecondProgram: ELSE RUN ThirdProgram
```

In this case, if the value of the variable Value% in the first program is equal to 1, the second program is executed; otherwise, the second program is skipped and the third program is executed.

You can also include EXEC within a program to run other programs; however, EXEC within a program must be followed by an END statement, although you can still use an END statement later in the program. For example:

```
.
.
.
80 EXEC TEXT.FILE:END
.
.
.
```

RUNNING ASSEMBLY LANGUAGE FROM BUSINESS BASIC

Business BASIC allows you to run *assembly language subroutines* from within a Business BASIC program. If you're not familiar with the term, *assembly language* is a programming language that is very close to the machine language of the Apple III. Assembly language is advantageous because its programs run very fast and take up very little memory space. However, unless you're an experienced assembly language programmer, it can be difficult to use.

Your Business BASIC master disk has a number of assembly language *subroutines* (small programs written in assembly language). These subroutines, called *external routines*, are stored in the files that have the .INV extension. Most of the .INV files have a file with an identical name and the extension .DOC. These are document files that explain what the subroutines do and how they're used. To read the .DOC files, just run them using the RUN command.

You can also write your own assembly language subroutines and use them with Business BASIC.

Using External Routines

There are four statements in Business BASIC that allow you to use external subroutines:

INVOKE

PERFORM

EXFN

EXFN%

INVOKE

INVOKE is used to load assembly language subroutines into Business BASIC. You type it, followed by the pathname of the external subroutine. For example,

```
INVOKE /BGRAF.INV
```

loads a subroutine named BGRAF.INV. You can also load a number of subroutines by separating their pathnames with commas:

```
INVOKE /BGRAF.INV, .D2/PRINT, .PROFILE/CRUNCH
```

INVOKE erases any subroutines that you previously loaded into memory. Using INVOKE without a pathname erases any subroutines, freeing up memory space for programs.

Passing Arguments

An external assembly language subroutine is in many ways similar to a Business BASIC subroutine that you call with a GOSUB statement. There is, however, one major difference. An *internal subroutine* (called with a GOSUB) has access to all the constants and variables within the program. However, because a subroutine you use with INVOKE is *external* to the Business BASIC program, you must let the subroutine know where the information to work on resides and where to put the new information it processes. This is done using an *argument list*. The external subroutine uses the argument list to communicate with the main Business BASIC program (the *calling program*).

There are two kinds of external subroutines that can be called from a Business BASIC program: procedures and functions. It might be helpful to think of the connection between the subroutines and the main program as a one-way and two-way street. An external *function* is a two-way street, taking data from the main program and returning data to it. A *procedure* is a one-way street, in that it takes data from the main program but doesn't return a result.

The exact arguments passed to and from an external subroutine vary depending on what the subroutine and the calling program require. (We'll explain using external subroutines in more detail in the next chapter.)

PERFORM

PERFORM executes the subroutine that you specify, provided it's been loaded into memory by the INVOKE statement. If the subroutine requires arguments, the argument list is enclosed in parentheses, with individual arguments separated by commas. For instance:

```
PERFORM Special (1234.67, Widgets)
```

In this example, "Special" is the name of the assembly language subroutine. The real number 1234.67 and the value of the variable Widgets is passed to the subroutine.

A real number or single variable argument is passed to an external subroutine by using the value or the name of the variable in the

argument list. If you're passing an integer or long integer to an external routine, the character that specifies the variable type must *precede* the argument. A “%” specifies an integer, and “&” specifies a long integer. Here's an example:

```
PERFORM Special (%Employees, &Population)
```

In this case, the value of the integer “%Employees” and the long integer “&Population” is passed to the subroutine.

You can also pass the *address* (location in memory) of an argument to a subroutine by using an “@” sign. For instance,

```
PERFORM Special (123.43, @Widgets)
```

passes the address of the variable Widgets to the routine.

EXFN.

EXFN. is similar to PERFORM, except that it's used with an external function, which returns a value.

The syntax for EXFN. is a bit different from the PERFORM statement. Here's an example:

```
120 PRINT EXFN. Interest (.015, %Period)
```

We'll assume that “Interest” is an external subroutine (function) that calculates the interest on a loan. This statement would pass the interest rate (.015) and the value of an integer variable named “%Period” to the function and print out the value that's returned from the function.

As you can see, you can use EXFN. as part of a Business BASIC statement. Also, just like PERFORM, you can use an “@” to specify the address of a variable.

EXFN%

As you might guess, EXFN% is similar to EXFN., except that the value returned to the Business BASIC program is an integer.

FINDING THE BUGS

There will be times when you spend a great deal of time typing in a Business BASIC program but get nothing but error messages when you

run it. The process of finding and correcting the mistakes is called *debugging*.

Business BASIC is very helpful in debugging, usually giving you specific error messages and line numbers where errors occurred. (See Appendix B for a full list of Business BASIC error messages.)

TRACE and NOTRACE

The TRACE command helps you isolate an error by telling you which line is being executed at any given moment. You start the process by typing TRACE at the Business BASIC prompt prior to running a program. From that point on a “#” will be printed followed by the number of the line that is being executed.

You can turn off the tracing by typing NOTRACE at the Business BASIC prompt.

Handling Errors — ON ERR and OFF ERR

When an error occurs in a running program, you receive an error message and the program stops, returning you to immediate mode. However, there will be times when you want a program to report the errors and somehow keep running.

The ON ERR statement allows a program to branch to a subroutine when an error occurs. Within the subroutine you can generate your own error message and execute any error-handling procedures that are necessary. When you are through, you can return to the main program. Here's an example:

```

10  REM Error Handling
20  ON ERR GOSUB 100
30  PRINT"Type a whole number on the keyboard"
40  INPUT X%
50  PRINT"You Typed ";X%
60  GOTO 10
70  END
100 REM Error Handling Subroutine
110 PRINT"You can only type a whole number, try again"
120 RESUME

```

This program lets you know if you type anything other than a whole number. If you enter anything other than an integer, the program branches to the subroutine starting at line 100. The RESUME statement returns the program back to the point where the error was found.

OFF ERR is used to cancel the last ON ERR statement, allowing you to branch to different error subroutines in different sections of the program.

OPTIMIZING PROGRAMS

One of the goals of writing a program is to have it run as fast as possible and occupy as little memory space as possible. You won't have to worry about optimizing short programs, but as you become proficient in Business BASIC and write longer and longer programs, you'll be concerned about conserving memory space.

Remember that some of the things you do to make programs run faster take more memory space, while some of the space conservation measures will make the program run slower.

Here are a few suggestions for making your programs run faster.

- Avoid using constants whenever possible. You should assign the value of a constant to a variable immediately (for example, Interest=.01525), and use the variable for the value throughout the program. This is important if you repeatedly use an integer value in a real expression. Business BASIC needs more time to convert the integer to a real number than to look up the value of the variable.
- Variables that are referenced often from within a program should be assigned a value as early as possible. Business BASIC keeps a sequential list of variables in its memory; each time the variable is referenced, it must look up its value by going through the list. If the value of an often used variable is at the top of the list, the program will find it faster.
- Put often used subroutines as near as possible to the beginning of the program. Each time Business BASIC encounters GOSUB or GOTO statements that reference a line number less than the current line number, it starts at the beginning and searches through the entire program until it finds the subroutine or line number it wants.

Saving Space

Here are a few hints for making your program fit into less memory space:

- Use INPUT instead of DATA statements whenever possible, since

the data stored in a DATA statement can take up a great deal of space.

- Use external data files instead of including the data in the program.
- Use arrays of integers instead of arrays of real numbers whenever possible.
- Assigning constants to variables speeds up the program and also saves space in most cases because the memory space for the constant is allocated only once rather than for each occurrence of the constant.
- Use the same variables as often as possible. If you've used a variable to store a temporary result that you no longer need, reuse it when you need a variable to store a new value. (If you're reassigning the value of a string variable, use a FRE statement to reorganize the unused memory space.)
- If you repeat the same series of calculations often, use a subroutine. Rewriting the same routine over and over again uses a great deal of memory space.
- Although it's sometimes handier to use "1" as the first index variable in an array, use the "0" element (for example A(0)) that all arrays have.
- Use short variable names.

Memory Space

When you need to save memory space, it's handy to know how much space the Apple III uses to store programs and data.

Storage space (like memory) is given in *bytes*. A 128K Apple III has a total of 131,072 bytes of storage space; a 256K system has 262,144. The Business BASIC interpreter and the operating system take up nearly 70K of space—leaving about 60K of memory space for you to use in a 128K system and about 192,000 in a 256K system. A 60K program would be quite long. (As you can see, if you have a 256K system, you don't have to worry too much about saving space. But remember that the program you write just might be run on a 128K system.) You can find out how much memory space you have left by typing **PRINT FRE**.

Here's a summary of how much space the various elements of Business BASIC use:

Constants. Constants use one byte per digit.

Numeric Variables. One byte is used for each character of the variable name. Four bytes are used for a real number, one byte for the

type, and one byte for "linking" (attaching) to memory. (Integers use two bytes for the value; long integers use eight bytes.)

Strings. Strings are stored in two parts: a three-byte *string descriptor*, describing the length of the string and where it is located in memory, and one byte for each character in the string.

String Variables. Stored as a three-byte string descriptor, string variables use one byte for each character of the string name, one byte for each character of the string, plus five additional bytes.

Arrays. Arrays use two link bytes, one type byte, one byte for each character of the array name, one byte for the number of dimensions, and two bytes per dimension. Each element of an integer array takes two bytes; each element of a real integer takes four bytes; each element of a long integer array eight bytes, and each element of a string array three bytes per character.

Creating Graphics And Sound

8

One of the Apple III's most important and impressive features is its ability to create a wide range of high-resolution graphics. You can create your own graphics from Business BASIC by using the .GRAFIX device driver and the library of assembly language graphics routines stored under the name BGRAF.INV on your Business BASIC disk. It isn't necessary for you to know anything about assembly language programming, since all you need to do is use the INVOKE and PERFORM statements to call assembly language subroutines. (If you need a review, see Chapter 7.)

The Apple III's .AUDIO driver also lets you create a wide range of tones that you can string together into "music." This capability is also contained in assembly language subroutines that you can call from Business BASIC. Later in this chapter, we'll take a close look at using the audio driver.

Before we get started, make sure you have the .GRAFIX and .AUDIO drivers installed on your Business BASIC working disk. If you don't, get out your master disk and see Chapter 4 for instructions on how to install a device driver.

You need a color video monitor to create the most eye-popping graphics, but don't despair if you don't own one. You can still create impressive graphics with a monochromatic display. Even if you don't have a color display, you can still create color graphics for use when you purchase one.

Although creating graphics is a simple matter from Business BASIC, using the advanced features of the graphics driver requires a lot of programming experience. In this chapter, we'll concentrate on the "basics." (For additional information, see Volume 2 of the *Apple Business BASIC Reference Manual*.)

APPLE III GRAPHICS

The Apple III has four graphics modes, as shown in Figure 8-1.

- Mode 0. This mode is 280 dots across by 192 dots high. You can only create black-and-white graphics in mode 0. (These graphics display in black-and-white on a color monitor.)
- Mode 1. This mode is also 280 by 192 dots; however, 16 colors are available, although there are some restrictions.
- Mode 2. This is the highest-resolution, black-and-white mode; it is 560 dots across by 192 dots high.
- Mode 3. This is the unlimited 16-color mode that is 140 dots across by 192 dots high.

The individual points of the Apple III graphics display are part of an *x-y coordinate system*. The x-coordinates are those on the horizontal axis and y-coordinates are points on the vertical axis. The *origin* is the dot in the lower left-hand corner and has the coordinates 0,0. Figure 8-2 shows the x-y coordinate system for mode 0.

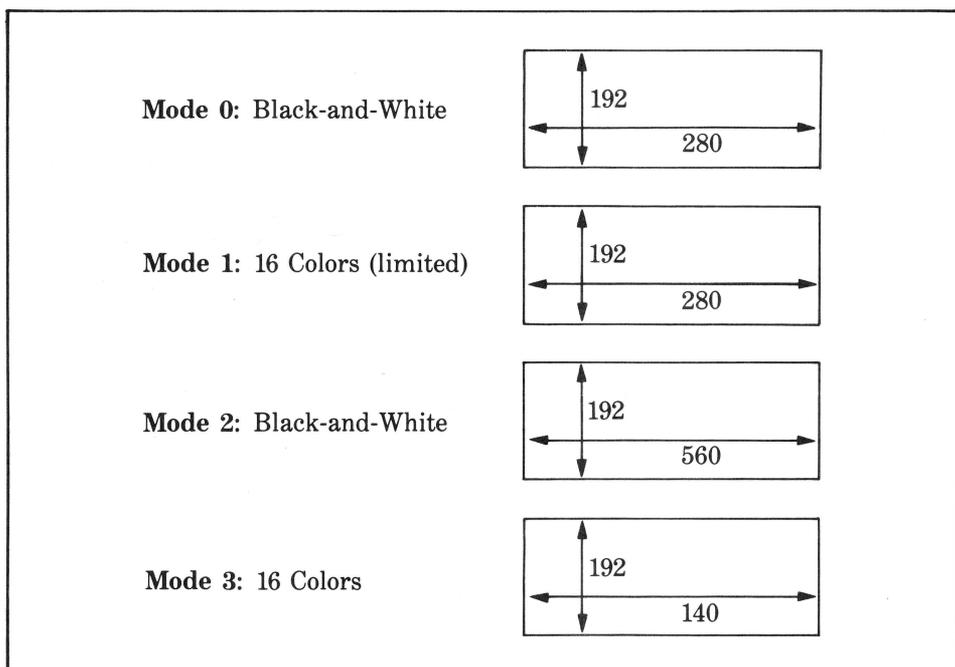


Figure 8-1. The four graphics modes

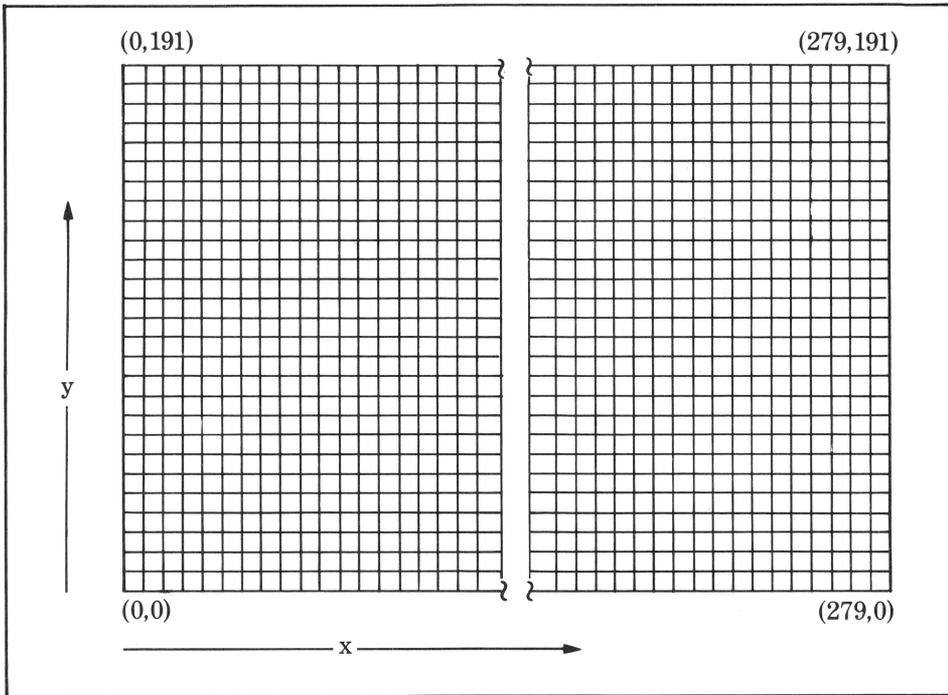


Figure 8-2. x-y coordinates in graphics mode 0

GRAPHICS DISPLAY BUFFERS

The graphics commands you'll be using don't operate directly on the video display. Instead, the graphics are "created" in special areas of memory called *display buffers*, which are set aside to hold the graphics data. Once an image has been stored in a display buffer, you need to execute a command to display the image on the screen.

You can use several display buffers; the exact number depends on the graphics mode you're using and whether your Apple III has 128K or 256K of RAM (random-access memory). You can display the contents of one buffer while a program is creating an image in another buffer.

APPLE III COLORS

Depending on which mode you use, the Apple III can produce graphics in 16 colors. To use the graphics routines, you'll need to refer to the colors by number (0-15). The color numbers are listed in Table 8-1.

Table 8-1. Graphics Mode Colors

Number	Color
0	Black
1	Magenta
2	Dark Blue
3	Purple
4	Dark Green
5	Gray 1
6	Medium Blue
7	Light Blue
8	Brown
9	Orange
10	Gray 2
11	Pink
12	Green
13	Yellow
14	Aqua
15	White

If you're using a monochrome video display in graphics modes 1 or 3, you'll see these colors as 16 sequential shades of gray ranging from black (0) to white (15). If you're using a monochrome video display in the black-and-white only modes (graphics mode 0 or 2), all the colors other than black (0) will be automatically converted to white.

Working With Color

When you create color graphics, you can select both the *pencolor* used for creating images (lines and dots) and the *fillcolor* that's used as the background. The *fillcolor* is also used to erase an image by "filling in" areas that you've already plotted.

Another term you should know is the *viewport*—the screen area on which you create graphics. When you first use the graphics driver, the viewport is automatically set to the entire screen area. However, you can change it by using the VIEWPORT procedure.

Although using the *pencolor* and *fillcolor* for plotting is the simplest way to create graphics with the Apple III, you can also use a *color table* to define what happens when you plot over a line or point you've already drawn. A *transfer option* allows you to change how the color table is used.

GRAPHICS ROUTINES

The BGRAF.INV module has a wide choice of procedures and functions you can invoke from Business BASIC to create graphics. These procedures and functions will be explained as we go along, and a detailed reference is included in Appendix F. Here is a list of the routines in BGRAF.INV:

- INITGRAFIX Resets graphics defaults.
- GRAFIXMODE Sets the mode and selects the display buffer.
- GRAFIXON Displays the currently used graphics buffer on the video display.
- PENCOLOR Selects the color used for plotting.
- FILLCOLOR Selects the erase color.
- SETCTAB Changes the color table.
- XFROPTION Transfer option.
- VIEWPORT Sets the graphics area of the screen.
- MOVETO Moves the cursor to the coordinates you specify.
- MOVEREL Moves the cursor relative to the current cursor position.
- DOTAT Draws a dot at the coordinate you specify.
- DOTREL Draws a dot relative to the cursor position.
- LINETO Draws a line from the current cursor position to the coordinates you specify.
- LINEREL Draws a line relative to the current cursor position.
- FILLPORT Erases the viewport.
- XYCOLOR A function that returns the color at the coordinates you specify.
- XLOC A function that returns the x-position of the cursor.
- YLOC A function that returns the y-position of the cursor.
- NEWFONT Changes the character font.
- SYSFONT Returns to normal type font.
- DRAWIMAGE Displays a previously created graphics image on the screen.
- GSAVE Saves a graphics image you've created to disk.

GLOAD Loads a previously created graphics image from memory.

RELEASE Releases graphics memory space for running non-graphic Business BASIC programs.

GETTING STARTED WITH .GRAFIX

Before you can create graphics, you'll have to start the .GRAFIX driver by opening a file. Use the OPEN# command and open a file like the following:

```
) OPEN#1, ".GRAFIX"
```

Remember, you can use any file number from 1 to 10, depending on whether you have any other files open in your Business BASIC program. Next, you'll need to invoke the graphics module by typing

```
) INVOKE "BGRAF.INV"
```

The disk drive light will come on for a few seconds and the graphics subroutines will be loaded into the Apple III's memory.

When you open the .GRAFIX driver, a number of default conditions are set as follows:

Graphics mode 0 (280 × 192, black-and-white)

Display buffer 1

Full screen viewport (x=0 to 279, y=0 to 191)

Cursor in the lower left-hand corner

Pencolor=15 (white)

Fillcolor=0 (black)

Normal color table

Normal transfer function

Standard text font

You can change any of these by using the procedures in the Business BASIC graphics module. If you wish, you can change the defaults by editing the .GRAFIX driver (see Chapter 4).

INITGRAFIX — Initialization

When you're working with graphics, you can at any time reset four of the graphics parameters to their default condition by typing

```
) PERFORM INITGRAFIX
```

Following the execution of INITGRAFIX, the viewport is set to the full screen, the cursor is positioned in the lower-left corner, and the color table and the transfer option are set to normal.

Choosing Mode and Buffer

The GRAFIXMODE procedure lets you change the graphics mode or the display buffer you're working with. Remember though, that since GRAFIXMODE is an external procedure (see Chapter 7), you'll have to "pass" the values of variables using arguments. If you wanted to change the graphics mode to 1 and use the second buffer, you could do something like the following:

```
)MODE=1
)BUFFER=2
)PERFORM GRAFIXMODE (%MODE, %BUFFER)
```

NOTE: All graphics procedures require the arguments to be integers. Therefore, you must precede a variable name with a percent sign.

Displaying Results

The GRAFIXON procedure displays the contents of the graphics buffer you've selected with GRAFIXMODE (or the default). Even after you've selected a new graphics mode and buffer, your Apple III screen will still display the previous screen until you type

```
)PERFORM GRAFIXON
```

Until you actually create an image, what you see is a screen that's nothing but the group of random dots that were in the graphics memory when you opened the graphics driver. You should make it standard operating procedure to use

```
)PERFORM FILLPORT
```

to fill in the viewport with the screen color you've selected. You can use FILLPORT anytime you want to erase a graphics screen and start anew.

Returning to Text Mode

Returning from a graphics mode to Business BASIC's normal text mode is simply a matter of typing

```
)TEXT
```

Although you won't see the command as you type the characters, as soon as you press RETURN you'll be returned to the text mode.

SETTING THE VIEWPORT

As we explained earlier, the viewport is the area on the screen in which graphics appear. It's similar to the WINDOW statement in Business BASIC. When you start the graphics driver, the default condition gives you a viewport that fills the entire screen and gives you (in mode 0) an x-axis (horizontal) that extends from 0 to 279 and a y-axis (vertical) that extends from 0 to 192.

You can change the dimensions of the viewport by using the VIEWPORT procedure. It has four arguments, which we'll call LEFT, RIGHT, BOTTOM, and TOP. Here's a short program that will generate a smaller viewport:

```

)LEFT=50
)RIGHT=200
)BOTTOM=40
)TOP=150
)PERFORM VIEWPORT (%LEFT,%RIGHT,%BOTTOM,%TOP)

```

This series of commands will set the left boundary of the viewport to x=50, the right boundary to x=200, the top boundary to y=150, and the bottom boundary to y=40. You can return the viewport to normal by typing

```

)PERFORM INITGRAFIX

```

CHOOSING THE PENCOLOR

To help you further understand how graphics are created, you must become acquainted with the *foreground* and *background*. The background is the screen you'll be "drawing" on. Normally it is black when you are in a black-and-white graphics mode. The foreground is the color you'll be "drawing" with. The PENCOLOR procedure is used to choose the foreground color. The color numbers used with PENCOLOR are the same as those listed in Table 8-1. In modes 0 and 2 (black-and-white only), all colors except 0 (black) are converted to white. If you're using a black-and-white video display with the color modes, you'll see the colors as varying shades of gray. The higher the color number, the lighter the shade of gray.

The default pencolor is 15 (white). You use the `PENCOLOR` procedure to change the pencolor. For instance,

```
)PERFORM PENCOLOR(15)
```

will change the pencolor to orange if you're in one of the color modes or white if you're using the black-and-white mode. You can change the pencolor at any time.

FILLCOLOR

As you might guess, the `FILLCOLOR` procedure sets up the color used for "filling in" (or erasing) shapes and areas that you've already created. Normally, the fillcolor is the same color as the background color.

For instance, in a color graphics mode,

```
)PERFORM FILLCOLOR(15)
```

will set the fillcolor to orange. To erase something you've already created, use the `FILLPORT` procedure to fill in the viewport with the fillcolor.

MOVING THE CURSOR

In order to create graphics, you need to move the cursor around the graphics screen. This can take some getting used to since you can't actually see an image until you use the `GRAFIXON` procedure. Whenever you enter one of the graphics modes, the cursor is at the lower left-hand corner of the graphics screen (the *origin*). There are two procedures that move the cursor without drawing anything, `MOVETO` and `MOVEREL`; two procedures that move the cursor and draw dots, `DOTAT` and `DOTREL`; and two procedures that move the cursor and draw lines, `LINETO` and `LINEREL`. Let's look at each of them.

MOVETO

The `MOVETO` procedure moves the cursor to the x-y coordinates you specify. It has two parameters: the x-coordinate and the y-coordinate. Here's an example:

```
)PERFORM MOVETO(100,120)
```

This moves the cursor to $x=100$ and $y=120$. Note that the cursor is moved to this position, *regardless* of where it is on the graphics screen. Although the cursor is at the new position, you can't see it (even if you use `PERFORM GRAFIXON`). Business BASIC's graphics cursor is always invisible.

MOVEREL

The `MOVEREL` procedure moves the cursor *relative* to its current position. It also has two parameters specifying how far to move the cursor in the x and y direction. Assuming the cursor is at $100,120$, you could move it to $x=80, y=160$ by typing

```
)PERFORM MOVEREL (%-20,%40)
```

DOTAT

The `DOTAT` (dot at) procedure is similar to `MOVETO` except it draws a dot at the final cursor position. (The dot is the color of `pencolor`.) Try this example:

```
)PERFORM DOTAT (%80,%130)
```

This places a dot at $x=80, y=130$. To see for yourself, type

```
)PERFORM GRAFIXON
```

You'll see a small dot on the screen. Type `TEXT` to return to the Business BASIC screen.

DOTREL

Like `MOVEREL`, the `DOTREL` procedure moves the cursor *relative* to the present cursor position. In addition, it draws a dot. Let's put a second dot on the screen:

```
)PERFORM DOTREL (%50,%-60)
```

This places a dot at $x=130, y=70$. (Use `PERFORM GRAFIXON` and `TEXT` to confirm that the dot is there and to return to the text screen.)

LINETO

The `LINETO` procedure draws a line from the present cursor posi-

tion to the new coordinates you specify. Let's draw a line from the dot we just drew to a point in the upper right-hand corner:

```
)PERFORM LINETO (%200,%170)
```

LINEREL

The LINEREL procedure moves the cursor to a position *relative* to the current cursor location and then draws a line between the old and new cursor position. Now let's draw a line back to a position in the lower left-hand area of the screen:

```
)PERFORM LINEREL (%-180,%-160)
```

SAVING AND RETRIEVING GRAPHICS _____

Once you have created a graphics image, you can save it in a Business BASIC file. Images saved in this way can be loaded from within any Business BASIC program.

GSAVE

GSAVE works almost exactly like Business BASIC's SAVE command. However, since it's part of the graphics module, you'll need to use it with the PERFORM command. For instance, to save the lines we created a few paragraphs ago, you can type

```
)PERFORM GSAVE. "DEMO.GRAPHIC"
```

The rules for naming your graphics file are the same as for naming any other Business BASIC file. When you type CAT, you'll see your file saved with the type FOTO. (All graphics are saved with the FOTO type.) GSAVE saves not only the graphics, but also all the parameters such as mode, color, and so forth.

GLOAD

To transfer a graphic image into the current graphics buffer you use the GLOAD command. But remember that if you're including a graphics image in the middle of a "standard" Business BASIC program, you'll

need to open a file for graphics and invoke the graphics module. Here's an example:

```

.
.
)340 OPEN#1, ".GRAFIX"
)350 INVOKE "BGRAF.INV"
)360 PERFORM GLOAD. "DEMO.GRAPHIC"
)370 PERFORM GRAFIXON
)380 FOR X=1 TO 1000
)390 NEXT X
)400 TEXT: PERFORM RELEASE
.
.
.

```

When used in the middle of a program, these lines would open a file for graphics, invoke the graphics module, load the demonstration graphic, display it for a few seconds, and then return to the text (nongraphics) mode.

RELEASE

If you're observant, you noticed that the last statement in the short program we just used was `PERFORM RELEASE`. This Business BASIC statement should be used when you're exiting the graphics mode. As you might guess, `RELEASE` "releases" memory space that is reserved for graphics.

Each time you change the graphics mode or the display buffer, additional memory space is set aside. Although the memory of your Apple III is large by personal computer standards (especially if you have a 256K machine), if you use lots of graphics and have a large program, you could eventually run out of space. Therefore, it's good programming practice to use `RELEASE` each time you leave the graphics mode.

NOTE: `RELEASE` "reclaims" graphics memory in steps. The first `RELEASE` statement reclaims 16K of buffer space, and each subsequent release reclaims 8K. If you were using the maximum amount of graphics memory (32K), you'd need to use `PERFORM RELEASE` three times.

Closing the Graphics Driver

If you leave the graphics mode and do not plan to return, you should close the file that contains the graphics driver. You could change the previous example to

```

.
.
.
)340 OPEN#1, ".GRAFIX"
)350 INVOKE "BGRAF.INV"
)360 PERFORM GLOAD. "DEMO.GRAPHIC"
)370 PERFORM GRAFIXON
)380 FOR X=1TO 1000
)390 NEXT X
)400 TEXT: PERFORM RELEASE
)410 CLOSE#1
.
.
.

```

CLOSE#1 in line 410 closes the graphics file. If you need to use graphics again from within the same program, you need to reopen the file and invoke the graphics driver once again. Doing so would restore the default graphics mode conditions.

Note that we still had to use RELEASE to reclaim the graphics memory for program use since closing the graphics file doesn't do that. You must, of course, use RELEASE *before* you close the file.

Remember that it isn't absolutely necessary to close the graphics file if you won't be doing additional graphics, but it's a good programming practice to do so. Any time you use RUN, LOAD, or NEW statements, all open files are automatically closed.

ADVANCED GRAPHICS PROCEDURES

Until now, we've shown you how to create simple dot and line graphics from Business BASIC. But there are a number of advanced procedures that allow you to customize how the graphics will look.

SETCTAB

When a line you've drawn crosses over another line in one of the color graphics modes, it's drawn *over* the line it crosses. (You don't have to worry about this in the black-and-white graphics modes since all lines will be black or white.) For example, if you draw a purple line, and then draw a brown line that crosses it, the brown line appears on top of the purple line. (The point where the lines intersect is changed to brown.) The SETCTAB procedure lets you set up a *color table* that specifies what should happen when a line of one color crosses a line of another color or when a dot of one color is plotted over a dot of another color. For

instance, you might want to have a new line crossing under another line and not change the original color where the lines intersect.

SETCTAB has three parameters:

- The source color (current pencolor or fillcolor).
- The screen color (color of the existing dot or line that will be plotted over).
- The color that will result when a dot or a line is plotted over a dot or line of a different color.

Since there are 16 source colors and 16 screen colors, you have a total of 256 color combinations.

If you want to have a yellow line (13) show medium blue (6) where it crosses an orange area (9), you could use something like the following:

```
)PERFORM SETCTAB(%13,%9,%6)
```

Here 13 is the color of the source, 9 is the color of the screen, and 6, the color of the result.

If you don't use the SETCTAB procedure, the color of the intersection where two lines cross will be the color of the most recently drawn line.

XFROPTION — Black and White

XFROPTION is similar to SETCTAB in that it specifies what happens when one color is plotted over another. However, because of its many options (and the strange things that can happen when multiple colors are plotted with it), XFROPTION is used with black-and-white graphics.

XFROPTION has only one argument: the number specifying the option. There are eight options (0-7), as shown in Table 8-2.

The PERFORM XFROPTION statement tells Business BASIC to use the transfer option you specify from that point on as you proceed with plotting. For instance,

```
)PERFORM XFROPTION(%1)
```

means that all plotting from that point on will use the overlay option. If you want to return to the normal replace option, you'd need to type

```
)PERFORM XFROPTION(%0)
```

Let's look at the options more closely.

Replace (0). This is the default option for graphics. When you plot a

Table 8-2. XFROPTION options

Option Number	Transfer Option	Result Color
0	Replace	Source color
1	Overlay	(Source color) OR (Screen color)
2	Invert	(Source color) XOR (Screen color)
3	Erase	(NOT Source color) AND (Screen color)
4	Inverse Replace	(NOT Source color)
5	Inverse Overlay	(NOT Source color) OR (Screen color)
6	Inverse Invert	(NOT Source color) XOR (Screen color)
7	Inverse Erase	(Source color) AND (Screen color)

line on the screen, it ignores whatever other color is on the screen and draws the line in the color you specify.

Overlay (1). Allows you to plot over an existing background or figure without erasing the background or figure.

Invert (2). Primarily used to plot white lines through a white or black area. White lines drawn through a black area will show up white, and white lines drawn through a white area will show up black. Plotting the same white line twice will erase the line.

Erase (3). A white line crossing a white area turns to black. (It's called "erase" because it is used to erase a white area.) All other combinations of lines and backgrounds have no effect.

Inverse Replace (4). Converts a white line to black and a black line to white, then plots over the existing screen.

Inverse Overlay (5). Converts a white line to black and a black line to white, then plots over the existing screen without erasing the existing figures or background.

Inverse Invert (6). Primarily used to plot black lines through a white or black area. Black lines through black areas show up white, and black lines through white areas show up black.

Inverse Erase (7). A black line will appear anywhere a line crosses a white area.

To get an accurate picture of the result, you need to work with the binary representation of the colors. In this case, the binary representation for black is 0000 0000, and white is 1111 1111. Notice that Table 8-2 expresses the resulting color in terms of a logical equation. If you combine the binary representation for the source color and the screen color

using the operations shown in Table 8-2, you will be able to predict what happens when you plot one color over another.

Admittedly, XFROPTION is a difficult procedure to understand. You'll seldom have any use for it unless you're an experienced programmer creating complicated graphics.

READING THE CURSOR LOCATION

The graphics module has three functions that let you read the location of the cursor and the color of the dot at the cursor location. Besides allowing you to keep track of the cursor's position and color while you're creating graphics, these functions are essential for programs such as games that require the computer to know where the cursor is.

Since they are *external functions*, you'll need to use the external function statement (EXFN%) in order to use them.

The function XYCOLOR returns the color of the dot at the current location of the cursor. For instance, try these statements:

```
)COLOR%=EXFN%.XYCOLOR
)PRINT COLOR%
```

The XLOC function returns the x-coordinate of the cursor; YLOC returns the y-coordinate. You can use statements like the following to find the coordinates:

```
)X%=EXFN%.XLOC:Y%=EXFN%.YLOC
)PRINT "THE CURSOR POSITION IS ";X%,";Y%
```

MIXING TEXT AND GRAPHICS

As you know, a graph without labels is essentially useless; you need to be able to print regular text with your graphics. That's easy to do in Business BASIC with PRINT#. Since we've already opened the .GRAFIX device driver as file number 1, PRINT#1 will allow you to print text on a graphics screen.

All you need do is move the cursor to where you want the text to appear and use PRINT#. Here's an example:

```
)PERFORM MOVETO(%50,%100)
)PRINT#1 "APPLE III GRAPHICS"
)PERFORM GRAFIXON
```

(Type **TEXT** to return to the text screen.) You'll notice that the size of the characters is considerably different from text mode characters. In fact, the size of text characters varies from graphics mode to graphics mode. Here's the maximum number of text characters you can display on one line in the various modes:

Mode 0 40 characters
 Mode 1 40 characters
 Mode 2 80 characters
 Mode 3 20 characters

Note that although **PRINT#** for graphics follows all the rules for a normal **PRINT** statement (you can use **TAB**, **SPC**, commas, and semicolons), it does *not* recognize the edge of the screen. If you use **PRINT#** for characters beyond the edge of the screen, they'll be lost. (To refresh your memory, in text mode letters are automatically wrapped around to the beginning of the next line when they reach the right edge of the screen.)

Text characters appear in the color set by the **PENCOLOR** procedure. We strongly suggest that you use white characters most of the time, since they're the most readable. Although you can use other colors, certain combinations are sometimes difficult to decipher.

BEYOND SIMPLE GRAPHICS — BIT ARRAYS _____

Up to now, we've created simple graphics using the graphics driver. With some careful planning, you can create impressive graphics using the procedures and functions described. However, there are still a number of graphics procedures not yet explained. Business BASIC lets you store images as *bit arrays*—the actual arrays of 1s and 0s that make up a "picture." The **.GRAFIX** driver has a number of procedures that let you work with bit arrays.

The **DRAWIMAGE** procedure displays a graphic image that you've previously created and stored in a bit array.

The **NEWFONT** procedure lets you change the type styles of text that you mix with your graphics by using a custom font you've stored as a bit array.

The **SYSFONT** procedure returns the graphics font to the standard character set.

NOTE: Even though storing an image as a bit array allows you to do

a number of advanced graphics, creating, storing, and using a bit array requires quite a bit of programming experience and an intimate knowledge of the binary and hexadecimal numbering systems. Since this book is designed for the average Apple III user, a discussion of bit arrays is beyond our scope. If you're interested in learning more, see Volume II of the *Apple Business BASIC Reference Manual*.

CREATING MUSIC WITH THE APPLE III _____

By this point, you're probably well aware of the "beep" that the Apple III produces when you make an error. However, your Apple III is capable of creating more than a single beep. You can control the volume, frequency, and duration of the tone to create music.

In order to produce tones, you'll need to have the .AUDIO device driver installed on your Business BASIC disk. (If it isn't installed, see Chapter 4.) You send data to the .AUDIO driver in the form of special characters. After the tone or succession of tones is generated by the speaker, the .AUDIO driver immediately returns control to the Business BASIC program. In that way, it differs from the .GRAFIX driver, which requires a TEXT command to return control to Business BASIC.

Audio Parameters

In order to produce music, you must send the audio driver four parameters:

Mode. This tells the driver that you're requesting a tone. The number you *must* use for the mode is 128.

Volume. How loud the tone will be. There are 64 volume levels ranging from 0 (volume off) to 63 (loud).

Count. The pitch of the tone. The range of count is 18383 to 100. The higher the count, the lower the pitch, giving you a range from 31 to 5000 Hz (cycles per second). Use this formula to determine the count:

$$\text{frequency} = \frac{50900}{\text{count}} \quad (18383 \text{ to } 100)$$

This produces a seven-octave range—from C three octaves below middle C to E-flat four octaves above middle C.

Time. The length of the tone. The range is from 0 (no tone) to 300

(about five seconds). Use the following formula to determine the duration of a tone in seconds:

$$\text{duration} = \frac{\text{time (0-300)}}{60}$$

Making Music

The technical details of how an audio driver produces tones are beyond the scope of this book. (For detailed information, see Apple's *Standard Device Driver's Manual*.) The audio driver *only* produces tones when it receives a six-character string, one character each for the mode and volume and two characters each for the count and time. If you're not an experienced programmer, converting the count and time to byte values can be difficult and time-consuming. Instead, here's a subroutine that you can plug into any program in which you want to create tones. It does all the necessary conversions for you.

```

.
.
.
)500 OPEN#1 AS OUTPUT, ".AUDIO":MODE%=128
)510 PRINT#1;CHR$(MODE%);CHR$(VOL%);
)520 PRINT#1;CHR$(CNT%-256*INT(CNT%/256));CHR$(INT(CNT%/256));
)530 PRINT#1;CHR$(TIME%-256*INT(TIME%/256));CHR$(INT(TIME%/256));
)540 CLOSE#1:RETURN

```

This subroutine makes all the necessary ASCII and other conversions so that you can enter VOL%, CNT%, and TIME% directly in your program and use a GOSUB statement to produce the tones. (We suggest you carefully copy this program. The semicolons are very important. Then SAVE it on your BASIC disk under a logical name. That way, every time you create a BASIC program in which you want to create tones, all you have to do is merge it into your program using LOAD. We used arbitrary line numbers for the subroutine, but you can choose your own. You can also use file numbers other than 1 if you wish.)

Using this subroutine to create music is a simple matter of using FOR-NEXT, DATA, and READ statements. Here's an example of a short program that runs through eight octaves of C:

```

10 VOL%=30:TIME%=10
20 FOR X=1 TO 8
30 READ CNT%
40 DATA 15564,7782,3891,1946,973,486,243,122
50 GOSUB 500
60 NEXT X
70 END
500 OPEN#1 AS OUTPUT, ".AUDIO":MODE%=128

```

```

510 PRINT#1;CHR$(MODE%);CHR$(VOL%);
520 PRINT#1;CHR$(CNT%-256*INT(CNT%/256));CHR$(INT(CNT%/256));
530 PRINT#1;CHR$(TIME%-256*INT(TIME%/256));CHR$(INT(TIME%/256));
540 CLOSE#1:RETURN

```

This program plays each tone with the same volume and duration, but you can also vary them. Here's the same program that varies the duration for a miniature piece that might be called "The Call of the Wild Programmer."

```

10 VOL%=30
20 FOR X=1 TO 8
30 READ TIME%,CNT%
40 DATA 25,15564,5,7782,15,3891,5,1946,30,973,20,486,1,243,100,122
50 GOSUB 500
60 NEXT X
70 END
500 OPEN#1 AS OUTPUT, ".AUDIO":MODE%=128
510 PRINT#1;CHR$(MODE%);CHR$(VOL%);
520 PRINT#1;CHR$(CNT%-256*INT(CNT%/256));CHR$(INT(CNT%/256));
530 PRINT#1;CHR$(TIME%-256*INT(TIME%/256));CHR$(INT(TIME%/256));
540 CLOSE#1:RETURN

```

As you can see, the possibilities are almost endless. You can vary the volume as well. If you're musically inclined, your Apple can become a versatile music maker.

Converting Count to Notes

You have to enter a COUNT value in order to produce tones with your Apple III. To do this, you have to know which count values correspond to standard notes. Table 8-3 shows the count values and the notes they correspond to.

Finally, if you want to create a certain frequency in Hz, you can use a variation of the formula we used earlier in this section:

$$\text{Count} = \frac{50900}{\text{Frequency}}$$

Table 8-3. Count Values for Creating Notes

Octave	1	2	3	4	5	6	7	8
B	8245	4122	2061	1031	515	258	129	
A#	8735	4368	2184	1092	546	273	136	
A	9255	4627	2314	1157	578	289	145	
G#	9805	4902	2451	1226	613	306	153	
G	10388	5194	2597	1298	649	325	162	
F#	11006	5503	2751	1376	688	344	172	
F	11660	5830	2915	1457	729	364	182	
E	12353	6177	3088	1544	772	386	193	
D#	13088	6544	3272	1636	818	409	204	102
D	13866	6933	3467	1733	867	433	217	108
C#	14691	7345	3673	1836	918	459	230	115
C	15564	7782	3891	1946	973	486	243	122
Middle C is 1946.								

Business BASIC

Quick Reference

A

This appendix is a complete reference to Business BASIC with commands and statements listed alphabetically. Although we suggest you thumb through it now, you'll find it most helpful when you're writing programs to jog your memory on particular commands and statements and to see how they're used.

In this appendix, we use a standard method to present the form of each statement and function in Business BASIC. Listed here are the punctuation, capitalization, and other conventions we used.

- | | |
|----------------|---|
| { } | Braces indicate repeatable items. One of the enclosed items must be present. (Braces don't appear in actual Business BASIC statements.) |
| [] | Brackets indicate that the enclosed items are optional. (Brackets don't appear in actual Business BASIC statements.) |
| | A vertical slash separates alternative items, any <i>one</i> of which can be used. |
| ... | Ellipses indicate that the preceding item can be repeated. (Ellipses don't appear in actual Business BASIC statements.) |
| Line numbers | All statements except those marked "immediate mode only" can be used in deferred execution with line numbers. |
| UPPERCASE | Uppercase words and letters must appear exactly as shown. |
| <i>Italics</i> | Generic terms are italicized. You'll have to supply the exact wording or value(s) according to the following generic definitions. |

The following italicized generic terms are used in statements and functions:

<i>const</i>	Any numeric or string constant.
<i>expr</i>	Any numeric string, relational, or Boolean constant, variable, or expression; any valid combination.
<i>expr\$</i>	Any string constant, variable, or expression.
<i>exprnm</i>	Any numeric constant, variable, or expression.
<i>filenum</i>	A specified disk file (1-10) created by an OPEN# statement.
<i>line</i>	Any BASIC program line number.
<i>memloc</i>	Any memory location.
<i>message</i>	Any text string enclosed in quotes.
<i>pathname</i>	The name of and "path" to a specified file. (See Chapter 3.)
<i>recnum</i>	A record number in a disk file.
<i>statementlist</i>	One or more Business BASIC statements.
<i>var</i>	Any numeric, constant, or string variable.
<i>varnm</i>	Any numeric variable name.
<i>var(sub)</i>	Any subscripted integer, numeric, or string variable.

BUSINESS BASIC STATEMENTS AND FUNCTIONS

ABS

Returns the absolute value of an arithmetic expression.

Format: ABS(*expr*)

Examples:

```
)PRINT ABS(23.67)
23.67
)PRINT ABS(-23.67)
23.67
```

Arithmetic Operators

Business BASIC has nine arithmetic operators. The *operands* can be real numbers, integers, or long integers (up to 19 digits in length). Long integers cannot be mixed with real numbers or integers.

Operator	Operation
+	Positive number
-	Negative number
^	Exponentiation
*	Multiplication
/	Division
MOD	Modulo*
DIV	Integer division*
+	Addition
-	Subtraction

*Used with long integers only.

Arrays

An ordered collection of individual variables. All the variables in an array must be of the same type. The *array name* is any legal variable name, with the last character the type of variable within the array as follows:

- % Integer
- & Long Integer
- \$ String

Each *element* of the array is numbered, with the first element numbered as 0. To access any element of an array, specify the name of the array followed by the *subscript* (the number of the element within the array).

An array can have one, two, three, or more dimensions, and the number of elements in any array is limited by the available memory. If you use an array without using a DIM statement (see DIM), Business BASIC creates an array with 11 elements per dimension, numbered 0-10.

Examples:

```

)PRINT WIDGETS%(8)

)MONTH$(7)="JULY"

)PRINT SALES(3,4)

```

ASC

Returns the ASCII value (in decimal) of the first character of a string expression.

Format: ASC(*expr*\$)

Example:

```
)PRINT ASC("APPLE")
65
```

ATN

Returns the arctangent of an arithmetic expression. The value returned is in radians.

Format: ATN(*expr*)

Example:

```
)PRINT ATN(.5643)
.513756
```

BUTTON

Returns a value depending on how the switch of a game paddle is set. BUTTON(0) and BUTTON(1) refer to the paddle connected to PORT B on the rear panel of the Apple III. BUTTON(2) and BUTTON(3) refer to the paddle connected to PORT A. In all cases, if the switch on the paddle is closed (pushed in), a value of 255 is returned; otherwise, a 0 is returned.

Format: BUTTON($0 \leq \textit{expr} \leq 3$)

Example:

```
100 IF BUTTON(0)=255 GOTO 200
```

CATALOG (or CAT)

Displays a list of files on the indicated volume or subdirectory.

Format: CAT[ALOG] [*pathname*]

Examples:

```
CAT
```

```
CAT .D2
```

CHAIN

Loads and runs the program or portion of a program you specify without clearing any variables or closing any files from the previous program.

Format: CHAIN *pathname* [,*line*]

CHAIN is primarily used for breaking a large program up into smaller pieces that can be run in sequence.

Example:

```
110 CHAIN .D2/LINK.FENCE
```

CHR\$

Returns the character corresponding to the ASCII code of the expression. Valid numbers are 0 through 255.

Format: CHR\$(*integer*)

Example:

```
)PRINT CHR$(120)
x
```

CLEAR

Sets all numeric variables to 0, clears all string variables, clears all BASIC pointers and stacks, and closes any open disk files.

Format: CLEAR

CLOSE and CLOSE#

Closes all open files or specified file number.

Format: CLOSE[# *filenum*]

Examples:

```
)780 CLOSE
```

```
)540 CLOSE#2
```

CONT

Resumes execution of a program that's previously been halted by STOP, END, or CONTROL C.

Format: CONT

[CONTROL] [5], [6], [7], [8], [9]

When entered from the *numeric keypad only*, the following combinations perform special functions:

[CONTROL 5] switches off the video screen. This allows programs to run faster since it takes a great deal of microprocessor time to "refresh" the screen. Pressing [CONTROL 5] a second time turns the screen back on.

[CONTROL 6] clears the typeahead buffer. For example, if you typed in a command while Business BASIC was running a program, the computer would complete what it's doing and then look at the typeahead buffer to see if you've typed in a command. If you change your mind before BASIC looks at the typeahead buffer, typing [CONTROL 6] will clear the buffer.

[CONTROL 7] turns output to the video screen off (although what was on the screen will remain there). A second [CONTROL 7] resumes output.

[CONTROL 8] makes the system ignore all special formatting ASCII control characters sent to the screen. A second [CONTROL 8] resumes normal operation.

[CONTROL 9] clears the screen buffer. A second [CONTROL 9] resumes normal operation.

CONV

Returns the real value of an arithmetic expression.

Format: CONV(*expr*)

Example:

```
)PRINT CONV(34.23)
34.23
```

CONV\$

Returns a string value of an expression.

Format: CONV\$(*expr*)

Example:

```
)PRINT CONV$(34.23)
34.23
```

CONV&

Returns the long integer value of an expression.

Format: CONV&(*expr*)

Example:

```
)PRINT CONV&(34.23)
34
```

CONV%

Returns an integer value of the expression, rounded off to the nearest whole number.

Format: CONV%(*expr*)

Example:

```
)PRINT CONV%(34.23)
34
```

COS

Returns the cosine of an angle given in radians.

Format: COS(*expr*)

Example:

```
)PRINT COS(2.435)
-.760579
```

CREATE

Creates directories, subdirectories, and text or data files. If you'll be using the file for random access, you can specify the length (in bytes) of

each *record* in the file. If the length isn't specified, each record is 512 bytes.

Format: CREATE *pathname*,CATALOG|TEXT|DATA [*expr*]

Examples:

```
100 CREATE BASIC/GAMES/Spacedodgers, DATA
```

```
110 CREATE Address.Book, TEXT, 200
```

DATA

Creates a list of values to be assigned in a READ statement.

Format: DATA *const* [*const*]..

Example:

```
110 DATA 23.45, "Flytrap", 23156
```

DEF FN

Defines functions to be used within programs.

Format: DEF FN *name* = *expr*

Example:

```
100 DEF FN Compute.Int=Period%/Princ*Int
```

DEL

Deletes specified program line(s).

Format: DEL *line* [TO|,|-*line*]

Examples:

```
DEL 110
```

```
DEL 100 TO 160
```

```
DEL 100,160
```

```
DEL 100-160
```

DELETE

Removes a subdirectory or file. A subdirectory can only be deleted after all its files have been deleted.

Format: DELETE *pathname*

Examples:

```
DELETE Intrstrate
```

```
DELETE /BASIC/Games
```

DIM

Allocates space for an array.

Format: DIM *var(integer[,integer]...)*

Examples:

```
100 DIM Widgets%(100,10,3)
```

```
110 DIM Receivables(200,200)
```

The maximum number of elements in an array depends on the available memory. If you refer to an array before using DIM, Business BASIC automatically creates an array with eleven elements (0-10) per dimension.

ELSE

Gives the program a command if the conditions of an IF-THEN statement are not met.

Format: ELSE *expr/line*

Example:

```
200 IF DAY$="Thursday" THEN GOSUB 600:ELSE GOTO 140
```

END

Stops program execution.

Format: END

Example:

```
99 END
```

EOF

End of file. When Business BASIC encounters an end of a file, an error occurs and the number of the file is assigned to the reserved variable EOF. You can keep the program from stopping by using a statement that tells Business BASIC what to do when it encounters an end of file. (See ON EOF.)

ERR

Error. When Business BASIC encounters an error, the code number of the error is assigned to the reserved variable ERR. You can "handle" an error (and keep the program from stopping) by using a statement that tells Business BASIC what to do when it encounters an error. (See ON ERR.)

EXFN.

Executes an assembly language function loaded by the INVOKE statement and returns a real number.

Format: EXFN.*pathname*[(*expr*|@*var*[{*expr*},@*var*}]]

Example:

```
)PRINT EXFN.Intrst(36)*.022
```

EXFN%

Executes an assembly language function loaded by the INVOKE statement and returns an integer.

Format: EXFN%.*pathname*[(*expr*|@*var*[{*expr*},@*var*}]]

Example:

```
)PRINT EXFN%.Inven(675)/Months%
```

EXP

Raises 2.718282 to the power indicated by the expression.

Format: EXP(*expr*)

Example:

```
)PRINT EXP(9)
8103.08
```

FOR-NEXT

FOR starts a loop that repeats a set of instructions. The variable is incremented or decremented until the given value is reached.

Format: FOR *varnm*=*exprnm* TO *exprnm* [STEP *exprnm*]

STEP is optional and increments (or decrements) the variable by the specified integer.

Example:

```

)10 FOR X=5 TO 30 STEP 5
)20 PRINT X-1;
)30 NEXT X
)40 END
)RUN
4
9
14
19
24
29

```

FRE

A reserved variable that stores the amount of remaining available memory (in bytes). Used to check the amount of space remaining in memory. Each time FRE is used, storage space for string variables is reorganized to recover space that isn't being used.

Format: FRE

Example:

```

)PRINT FRE
)196051

```

The amount of free memory depends on the size of the Apple III's memory and the program in memory.

GET

Assigns a character or number from the keyboard to a variable.

Format: GET *var*

Example:

```

)10 PRINT "Press a key"
)20 GET KEY$
)30 PRINT "You pressed ";KEY$
)40 END
)RUN
Press a key
You pressed D

```

GOSUB

Causes the program to branch to the indicated line. When RETURN is encountered, the program returns to the line number following the GOSUB.

Format: GOSUB *line*

Example:

```
80 IF Payment<100.00 THEN GOSUB 200
```

GOTO

Causes the program to branch to the indicated line.

Format: GOTO *line*

Example:

```
100 IF X%=10 THEN GOTO 140
```

HEX\$

Returns a four-character string that's the hexadecimal (base 16) equivalent of the expression.

Format: HEX\$(*expr*)

Example:

```
PRINT HEX$(726)
02D6
```

The range of numbers which can be converted to hexadecimal is -65535 to 65535.

HOME

Immediate mode only. Clears the screen and moves the cursor to the upper left-hand corner.

Format: HOME

HPOS

Sets the horizontal position (column) of the cursor in the range of 0 to 255. (See VPOS.)

Format: HPOS=*exprnm*

Example:

```
100 HPOS=40
```

IF-GOTO

Causes the program to conditionally branch to a specified line number if the specified conditions are true. Identical to using a combination of IF-THEN and GOTO statements.

Format: IF *expr* GOTO *line*

Example:

```
40 IF MOUSETRAP%(2,5)=2500 THEN GOTO 100
```

IF-THEN

Causes the program to execute given instructions if specified conditions are true.

Format: IF *expr* THEN *statement*

Example:

```
200 IF Answer$="Y" THEN PRINT "That's all folks!":END
```

IMAGE

Allows you to specify a format to be used in a PRINT USING statement. IMAGE is optional and allows you to use the same format several times in a program.

Format: (See Chapter 7.)

Example:

```
20 IMAGE 5#.3#,80C
100 PRINT USING 20; 2357.6432, "The Apple III"
```

INDENT

A reserved variable that defines the number of spaces to indent a FOR-NEXT loop. If you don't use an INDENT statement, the default value is 2.

Format: INDENT=*exprnm*

Example:

```
) INDENT=5
```

INPUT

Accepts input from the keyboard and assigns the value(s) to indicated variables.

Format: INPUT ["*prompt*",;]*var*[,*var*]

Example:

```
50 INPUT "Enter Today's Date (MM/DD/YY) ";Date$
```

INPUT#

Reads a line of text for each variable in a list. You define the file you're reading from by specifying a file number from 1 to 10. (The file must have previously been opened with the OPEN# statement.) In a random-access file, you can specify the record number to begin reading.

Format: INPUT#*filenum*[,*recnum*];*var*{,*var*}]

Examples:

```
40 INPUT#1,54;Score$
```

```
230 INPUT#2; A%,B,C$
```

INSTR

Searches for a substring within a string and returns a number representing the position of the first character of the substring.

Format: INSTR(*expr*\$,*expr*\$[,*expr*])

Example:

```
)PRINT INSTR("Business BASIC", "nes")
5
)PRINT INSTR("Business BASIC", "nes", 6)
0
```

The optional number tells INSTR at which character position to start searching. If Business BASIC doesn't find the string you specify, it returns a 0.

INT

Returns the largest whole number less than or equal to the value of the argument.

Format: INT(*expr*)

Examples:

```
)PRINT INT(23.499)
23
)PRINT INT(23.5)
23
```

INTEGER

Any positive or negative number without a decimal point in the range of -32768 to +32767.

INVERSE

Sets a reverse screen (black letters on a white background) for all subsequent characters. (See NORMAL.)

Format: INVERSE

Example:

```
100 INVERSE:PRINT "Inverse Text":NORMAL
```

INVOKE

Loads assembly language subroutines into memory.

Format: INVOKE *pathname* [{,*pathname*}]

Example:

```
) INVOKE BGRAF. INV
```

KBD

A reserved variable that contains the ASCII value of the last key pressed.

LEFT\$

Returns a string (of the specified length) that's composed of the leftmost characters of a string expression.

Format: LEFT\$(*expr\$,exprnm*)

Example:

```
) PRINT LEFT$("Business BASIC",3)
Bus
```

LEN

Returns an integer equal to the length of a string expression.

Format: LEN(*expr\$*)

Example:

```
) PRINT LEN("Business BASIC")
14
```

The maximum string length is 255 characters.

LET

Assigns the value of an expression to a variable name.

Format: [LET]

Example:

```
) LET Days%=263
```

```
) Days%=263
```

Since LET is optional, these two expressions are equivalent.

LIST

Lists (on the output device) the current program (or specified part of the program) in memory.

Format: LIST [*line*] [TO|,|- [*line*]]

Examples:

LIST

LIST 20 TO 100

LIST 20,100

LIST 20-100

LIST 20-

LIST -100

LOAD

Loads a BASIC program from a disk file into memory.

Format: LOAD *pathname*

Example:

LOAD /BASIC/Games/Spaceblaster

LOCK

Immediate mode only. "Locks" a file so that it can't be accessed or deleted. (UNLOCK reverses the action.)

Format: LOCK *pathname*

Example:

)LOCK .D2/Secret.File

LOG

Returns the natural logarithm of an expression.

Format: LOG(*expr*)

Example:

```
)PRINT LOG(.23145/6.3)
-3.30394
```

LONG INTEGER

Integers up to 19 digits long. Long integer variables must end with & (ampersand) and can be in the range from -9223372036854775808 to 9223372036854775807.

MID\$

Returns a substring from a given string. The first integer specifies the position of the first character to be returned and the second (optional) the length of the substring.

Format: MID\$(*expr*\$,*exprnm*[,*exprnm*])

Examples:

```
)PRINT MID$("Business BASIC",5)
ness BASIC
)PRINT MID$("Business BASIC",5,6)
ness B
```

NEW

Immediate mode only. Erases the current program and all its variables from the Apple III memory and closes all open files.

Format: NEW

NORMAL

Returns the video display mode to a "normal" (white on black) display after an INVERSE command.

Format: NORMAL

NOTRACE

Immediate mode only. Cancels TRACE. (See TRACE.)

Format: NOTRACE

OFF EOF#

Cancels an ON EOF# statement.

Format: OFF EOF#

Example:

```
100 ON EOF#2 PRINT "END OF FILE":GOTO 300
200 OFF EOF#2
```

ON EOF#

Executes the given statement if BASIC reads past the end of a specified file.

Format: ON EOF# *filenum statementlist*

Example:

```
100 ON EOF#3 PRINT "OUT OF DATA":GOSUB 300
```

ON ERR

Executes the given statement if BASIC encounters an error. Normally used to branch to an error-handling routine.

Format: ON ERR *statementlist*

Example:

```
20 ON ERR GOTO 670
```

OFF ERR

Cancels the most recent ON ERR statement.

Format: OFF ERR

Example:

```
20 ON ERR GOTO 670
100 OFF ERR
```

ON KBD

Instructs BASIC to execute a list of statements when any key on the keyboard is pressed.

Format: ON KBD *statementlist*

Example:

```
60 ON KBD GOTO 100
```

OFF KBD

Cancels the last ON KBD command.

Format: OFF KBD

Example:

```
60 ON KBD GOTO 100
```

```
200 OFF KBD
```

ON-GOSUB

Used to specify a subroutine location the program should branch to when it encounters a specified value.

Format: ON *expr* GOSUB *line* [*,line*]..

Example:

```
10 ON Days% GOSUB 100,200,300,400,500,600,700
```

In this example, the program will branch to 100 if Days% is 1; 200 if it's 2, and so forth.

ON-GOTO

Works exactly like ON-GOSUB, except the program branches to the line number you specify.

Format: ON *expr* GOTO *line* [*,line*]..

Example:

```
230 ON Days% GOTO 100,200,300,400,500,600,700
```

OPEN#

Opens specified file(s) for access.

Format: OPEN#*filenum*[AS INPUT|AS OUTPUT|AS EXTENSION], *pathname*[,*exprnm*]

Examples:

```

100 OPEN#1,"Bearight"
150 OPEN#2,"Bearleft"
200 OPEN#3 AS INPUT, ".CONSOLE"
250 OPEN#4 AS OUTPUT, ".PRINTER"

```

You can use any number from 1 to 10 as the file reference number. As many as ten files can be open at the same time.

OUTPUT#

Directs all screen output to the file you specify.

Format: OUTPUT#*filenum*

Example:

```

100 OUTPUT #1

```

Typing OUTPUT#0 returns screen output to the video display.

OUTREC

A reserved variable that contains the maximum length of lines output when using LIST. The value of OUTREC must be greater than the value of INDENT (see INDENT).

Example:

```

100 OUTREC=80

```

PDL

Returns the vertical and horizontal position (0-255,0-255) of a joystick connected to PORTS A or B on the rear case of the Apple III.

Format: PDL(0<=*expr*<=3)

Example:

```
20 VPOS=(PDL(0)/10):HFOS=(PDL(1)/30)
```

This example positions the cursor within the visible screen area according to the setting of the paddles.

PERFORM

Executes an assembly language procedure that you've previously loaded with an INVOKE statement.

Format: PERFORM *pathname*[(*expr*|@*memloc*[{*expr*|@*memloc*}]])]

Example:

```
120 PERFORM Interest(Period%,Prin,@X)
```

The *argument list* passes data to the procedure. The "@" passes the *address* of the variable to the procedure.

POP

Allows you to exit from one level of nested subroutines. When Business BASIC executes a POP statement, the current return address pointed to by the program stack pointer is removed from the stack. In the following example, POP in line 320 allows the program to branch back to line 60 instead of returning to the subroutine at line 220. If POP isn't used, the program will execute an infinite loop.

Format: POP

Example:

```
50  GOSUB 200
60  PRINT"Program Finished":END
200 PRINT"Subroutine 200"
210 PRINT"Going to subroutine 300"
215  GOSUB 300
220  REM This line is never executed"
230  GOTO 200
300  "Subroutine 300"
310  PRINT"Using POP to avoid return to line 220"
320  POP
330  REM Line 320 removes the pointer to line 220
340  RETURN
```

PREFIX\$

A reserved variable that contains the most recently assigned path-name prefix.

Example:

```
)PRINT PREFIX$
/BASIC/
```

PRINT

Displays characters on the output device.

Format: PRINT {[,;][*expr*]} [,;]

Examples:

```
PRINT Rate%
PRINT Rate%/3*12.093
PRINT "Message"
PRINT "Message";
PRINT "Message",
```

PRINT USING

Same as PRINT except that it formats the text using format specification characters.

Format: (See Chapter 7.)

Example:

```
)PRINT USING "5#.2#";765.569
765.57
```

PRINT#

Sends ASCII data to the text file you specify. (The file must have been opened by the OPEN# statement.)

Format: PRINT#*filenum* [,*recnum*][;*expr*[...;*expr*]][;]

Example:

```
430 PRINT#1;"This is ASCII text"
```

In a random-access file, you can also specify the record number the text should be written to.

Example:

```
430 PRINT#1,150;"This is ASCII text"
```

PRINT# USING

Same as PRINT# except it formats the data sent to the specified file.

Format: (See Chapter 7.)

Example:

```
300 PRINT#1 USING "BOC";"This text is centered"
```

Like PRINT#, you can also specify the record number in a random-access file.

Example:

```
300 PRINT#1,30 USING "BOC";"This is centered text"
```

READ

Assigns values to variables from corresponding values in a DATA statement.

Format: READ *var* [{*var*}]..

Example:

```
50 READ Rate,Period%,Principle
```

READ#

Retrieves information from the data file you specify. (The file must have been opened by the OPEN# statement.) You can also specify the record number in a random-access file.

Format: READ# *filenum* [,*recnum*] [*var* [{*var*}]]

Example:

```
300 READ#1, Message$,Employees%
```

REAL NUMBERS

Any positive or negative number in the range of $-1.7E38$ to $+1.7E38$.

REC

Returns the number of the record being accessed in the specified file.

Format: REC

Example:

```
100 PRINT REC
```

REM

Identifies a remark. Anything after REM is ignored by the program.

Format: REM *message*

Example:

```
100 REM This line is ignored by the program
```

RENAME

Immediate mode only. Changes the names of volumes, subdirectories, and local files.

Format: RENAME *old pathname, new pathname*

Example:

```
) RENAME OLDFILE, NEWFILE
```

RESTORE

Lets you read the same data more than once by telling the Apple III to move its internal pointer back to the beginning of the list of data.

Format: RESTORE

Example:

```
50 READ A$, B%, C
60 DATA "Nonsense", 2387, 98653.67
70 RESTORE
80 READ X$, Y%, Z
```

RESUME

Restarts program execution after an error-handling subroutine.

Format: RESUME

Example:

```

10  ON ERR GOSUB 300
20  INPUT "Type a whole number";NMBR%
30  PRINT "You typed ";NMBR%
40  END
300  REM This is an error handling subroutine
310  PRINT "You Goofed!!! Try again!"
320  RESUME

```

RETURN

Returns program execution to the statement after the last GOSUB executed.

Format: RETURN

Example:

```

50  GOSUB 200
60  PRINT "Returned from subroutine"
70  END
200  REM This is the subroutine
.
.
.
250  REM This is the end of the subroutine
260  RETURN

```

RIGHT\$

Returns a string of specified length that's composed of the rightmost characters of a given string expression.

Format: RIGHT\$(*expr\$,exprnm*)

Example:

```

)PRINT RIGHT$("Business BASIC",6)
BASIC

```

RND

Generates a random positive number less than 1.

Format: RND(*exprnm*)

Examples:

```

)PRINT RND(1)
.894111
)PRINT RND(2)
.324093

```

As long as the argument is greater than zero, RND generates a different random number each time you use it. RND(0) generates the same random number every time, although the number will differ each time you boot the system.

RUN

Immediate mode only. Starts execution of a program.

Format: RUN [*pathname[,line]]*[[*line*]]

Examples:

```

RUN /BASIC/Games/Spaceblaster
RUN /BASIC/Games/Spaceblaster,300

```

If you use a number, Business BASIC starts running the program at that line.

SAVE

Writes a copy of the current program in memory to a disk file.

Format: SAVE *pathname*

Examples:

```

SAVE Interest.Calc
SAVE /BASIC/Games/Spaceball

```

SCALE

Used with PRINT USING to shift the decimal point of a displayed value left or right.

Format: SCALE(*varnm,exprnm*)

Example:

```
100 PRINT USING "$10#.##";SCALE(-3,Income)
```

This example moves the decimal point of the variable "Income" three places to the left.

SGN

Returns -1 if the expression is negative, 0 if it's 0, and 1 if it's positive.

Format: SGN(*expr*)

Examples:

```
)PRINT SGN(345.34-675.23)
-1
)PRINT SGN(0)
0
)PRINT SGN(-2315.67+4000.0021)
1
```

SIN

Returns the sine of an angle given in radians.

Format: SIN(*expr*)

Example:

```
)PRINT SIN(8.3425)
.883029
```

SPC

Used with PRINT to define the number of spaces before the next character is printed.

Format: SPC(*expr*)

Example:

```
)PRINT "The";SPC(4);"Apple";SPC(4);"III"
The    Apple    III
```

SQR

Returns the positive square root of an expression.

Format: SQR(*expr*)

Example:

```
>PRINT SQR(72651)
269.538
```

STEP

Lets you increment or decrement steps in a FOR-NEXT loop.

Format: STEP(*exprnm*)

Example:

```
10 FOR X=2 TO 20 STEP 2
20 PRINT X
30 NEXT X
40 END
```

STOP

Halts program execution and returns to immediate execution.

Format: STOP

STR\$

Returns the value of an expression as a string.

Format: STR\$(*expr*)

Example:

```
>PRINT STR$(9823.426)

9823.426
```

SUB\$

Lets you replace any part of a string with a substring.

Format: SUB\$(*var,expr[,expr]*)=*expr*\$

Example:

```

)10 STRG$="SOFTWARE"
)20 SUB$(STRG$,1)="HARD"
)30 PRINT STRG$
)40 END
)RUN
HARDWARE

```

Note that in line 20, the "1" indicates the digit position in the original string where the new string is placed.

SWAP

Exchanges the value stored in two variables.

Format: SWAP *var,var*

Example:

```

)10 A$="Alpha":B$="Beta"
)20 SWAP A$,B$
)30 PRINT A$,B$
)40 END
)RUN
Beta           Alpha

```

TAB

Used in PRINT statements to define the number of spaces from the left margin for printing.

Format: TAB(*exprnm*)

Example:

```

)PRINT TAB(8) "HERE"
      HERE

```

TAN

Returns the tangent of an angle given in radians.

Format: TAN(*expr*)

Example:

```

)PRINT TAN(9.234)
-.193127

```

TEN

Returns the decimal equivalent of the last four hexadecimal numbers of a string expression.

Format: TEN(*expr*%)

Example:

```
PRINT TEN("4DFF")
19967
```

TEXT

Clears any text or graphics modes in use and sets the display to the normal full-screen text mode.

Format: TEXT

TRACE

Immediate mode only. Prints a “#” followed by the number of each program line as it executes. NOTRACE turns it off.

Format: TRACE

TYP

Tells BASIC what type of data will be read from a file.

Format: TYP(*filenum*)

For a data file, TYP returns the following values:

<i>filenum</i>	Meaning
0	Undetermined
1	Real
2	Integer
3	Long Integer
4	String
5	End of file

For a text file, TYP always returns an 8.

Example:

```
100 ON TYP(8) GOTO 300
```

UNLOCK

Immediate mode only. Unlocks a locked file so that it can be accessed. (See LOCK.)

Format: UNLOCK *pathname*

Example:

```
)UNLOCK .D2/MOUSE.GAME
```

VAL

Evaluates a string expression and returns the value as a real number or integer.

Format: VAL(*expr*\$)

Examples:

```
)PRINT VAL("2345")
```

```
2345
```

VPOS

Sets the vertical position of the cursor. (See HPOS.)

Format: VPOS=*exprn*

Example:

```
10 VPOS=12
```

The valid range is 0 to 255.

WINDOW

Sets the position and size of the text window.

Format: WINDOW *exprnm,exprnm* TO *exprnm,exprnm*

Example:

```
)WINDOW 23,10 TO 70,25
```

The first pair of coordinates specifies the upper left-hand corner of the text window; the second pair, the lower right-hand corner. In this example

- 23 is the starting column number of the window.
- 10 is the starting row number.
- 70 is the ending column number.
- 25 is the ending row number.

WRITE#

Sends data to the data file you specify. (The file must have been opened by the OPEN# statement.) You can also specify the record number for a random-access file.

Format: WRITE# *filename* [, *recnum*] [*expr* [{*expr*}]]

Examples:

```
50 WRITE#1; SALES
```

```
60 WRITE#1, 200; SALES
```


Business BASIC Error Messages

B

Apple III Business BASIC handles errors in one of two ways:

1. If an ON ERR statement was executed, an error condition causes the statement referred to in the ON ERR statement to be executed.
2. If an ON ERR statement was not executed or an OFF ERR was executed, an error condition causes the program to stop, an error message is displayed on the screen, and the Business BASIC prompt reappears.

The following pages contain a list of all the possible error messages you might see and an explanation of what they mean.

?BAD PATH ERROR

Business BASIC was unable to find the file you specified because it either doesn't exist or you entered it incorrectly.

?BAD SUBSCRIPT ERROR

You attempted to reference an array element that's outside the bounds of the array.

?CAN'T CONTINUE ERROR

You attempted to use the CONT (continue) statement after modifying a program. You can only use CONT if the program is unchanged; however, you may modify the values of variables.

?DATA WON'T FIT ERROR

You attempted to write more data to a record in a disk file than would fit.

?DEVICE NOT FOUND ERROR

There are three possible reasons for this error:

1. The device isn't connected.
2. The system isn't configured for the device.
3. You entered the device name incorrectly.

?DISK FULL ERROR

There is no space left on the disk you specified.

?DIVISION BY ZERO ERROR

Although it is not illegal to divide a number by zero, the result (infinity) is too large to be stored in memory.

?DUPLICATE FILE ERROR

You attempted to create or rename a file using the name of an existing file.

?EXTRA IGNORED

You supplied more values than an INPUT statement called for.

?FILE LOCKED ERROR

You attempted to modify a file that was locked.

?FILE NOT FOUND ERROR

You attempted to access a file that doesn't exist or you entered the name of the file incorrectly.

?FILE NOT OPEN ERROR

You attempted to access a file before it was opened with an OPEN statement.

?FILE TOO LARGE ERROR

The file is larger than the limits set for Apple III files.

?FILES BUSY ERROR

You attempted to delete or rename a file while it was open.

?FORMULA TOO COMPLEX ERROR

There are two possible reasons for this error:

1. Parentheses are nested more than 14 deep.
2. You attempted to evaluate an arithmetic expression where more than 14 operations had to be defined due to rules of precedence.

?ILLEGAL DIRECT ERROR

You attempted to use statements in immediate mode that can only be used for deferred execution. The statements that can only be used for deferred execution are

INPUT	ON ERR
DEF FN	ON KBD
GET	ON EOF#
RESUME	

?ILLEGAL QUANTITY ERROR

The parameter passed to a function was out of range. Some possible reasons for this error are

1. You used a negative array subscript (for example, WIDGETS(-3)).
2. You used the LOG function with a negative or zero argument.
3. You used the SQR function with a negative argument.
4. You used one of the following statements with an expression whose value was outside the allowable range:

MID\$	SPC	HEX\$
LEFT\$	WINDOW	TEN
RIGHT\$	TAB	INSTR
VPOS	SUB\$	SCALE
HPOS	CHR\$	ON...GOTO

5. You opened a file with a record length less than 3.
6. You specified a file number less than 1 or greater than 10.
7. You used a repeat value greater than 255 in a PRINT (#) USING statement or an IMAGE specification.

?INVOKE ERROR

You attempted to INVOKE a non-invokeable (non-assembly language) file.

?I/O ERROR

Input/Output to or from a peripheral failed, with either a mechanical or electrical problem causing a loss of data. Check all connections. If you still have problems, see your dealer.

?NEXT WITHOUT FOR ERROR

There are three possible reasons for this error:

1. You nested loops improperly. The control variables in a NEXT statement must be listed in the reverse order from the way they were encountered in FOR statements.
2. The control variable you specified in a NEXT statement doesn't correspond to the variable in a FOR statement.
3. You attempted to execute a NEXT statement before a FOR statement.

?NOT SOS ERROR

The disk you attempted to access was not written in Apple III SOS format.

?OUT OF DATA ERROR

There are two possible reasons for this error:

1. A READ statement was executed when all the data elements in a DATA statement had already been read.
2. A READ# statement encountered the end of the file or a WRITE# statement attempted to write a field that was larger than the file's record size.

?OUT OF MEMORY ERROR

There are four possible reasons for this error:

1. There isn't any memory available for a file buffer.
2. Your program is too large for available memory.
3. An invoked file won't fit into available memory.
4. Too much space was used by the variables.

?OVERFLOW ERROR

The result of a calculation was too large (larger than 1.7E38) to be represented in Business BASIC.

?PATH NOT FOUND ERROR

Part of the pathname you specified was invalid; either it doesn't exist or you entered it incorrectly.

?RANGE ERROR

You specified an illegal range in a DEL or LIST statement.

?REDIM ERROR

After you defined an array, you used another DIM statement to define the same array.

?REENTER

The value given for INPUT is the wrong type.

?RETURN WITHOUT GOSUB ERROR

More RETURN or POP statements were executed than GOSUB statements.

?SOS CALL ERROR

An error that Business BASIC doesn't recognize occurred within the operating system.

?STACK OVERFLOW ERROR

There are three possible reasons for this error:

1. You nested FOR-NEXT loops more than 9 deep.
2. You nested subroutines more than 23 deep.
3. More than 27 subroutines were entered from ON KBD statements prior to a RETURN.

?STRING TOO LONG ERROR

The string you entered is more than 255 characters long.

?SYNTAX ERROR

This error can be caused by any of the following:

1. Missing parenthesis in an expression.
2. Illegal character in a statement.
3. ON not followed by GOTO or GOSUB.
4. IF not followed by THEN or GOTO.
5. Arithmetic operation on a string.
6. A digit as the first character of a variable name.
7. You attempted to use something other than a real number for a user-defined function.
8. Variable name over 64 characters long.
9. Bad specification in an IMAGE specification.
10. Bad AS option for OPEN#.
11. Bad operator.
12. Following DEL with something other than a digit.
13. Anything that's not syntactically correct.

?TYPE MISMATCH ERROR

This error can be caused by any of the following:

1. The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa.
2. A function which expected a string argument was given a numeric one, or vice versa.
3. Exponentiation with long integers.

4. You specified a non-BASIC file when the program expected a BASIC file.
5. You used a non-real variable when a real was expected.
6. You used strings in an IF-THEN statement.
7. You specified an incorrect IMAGE specification for a string or number in a PRINT (#) USING statement.
8. You used READ# to read numeric data when the data was a string, or vice versa.

?UNDEF'D FUNCTION ERROR

You attempted to reference a user-defined function that was never defined.

?UNDEF'D STATEMENT ERROR

There are three possible reasons for this error:

1. You attempted to branch to a line number that doesn't exist from a GOTO, GOSUB or IF-THEN statement.
2. A PRINT USING statement refers to an IMAGE statement that doesn't exist.
3. IMAGE isn't the first statement in a line.

?VOLUME NOT FOUND ERROR

The volume name you specified in an input/output statement doesn't exist.

?WRITE PROTECT ERROR

The disk's write-protect notch is covered.

BUSINESS BASIC ERROR CODES_____

When Business BASIC detects an error, the reserved variable ERR will contain an error code number. You can see the code by typing

PRINT ERR

Here's what the codes mean:

- 1 NEXT WITHOUT FOR
- 2 SYNTAX
- 3 RETURN WITHOUT GOSUB
- 4 OUT OF DATA
- 5 ILLEGAL QUANTITY
- 6 OVERFLOW
- 7 OUT OF MEMORY
- 8 UNDEFINED STATEMENT
- 9 BAD SUBSCRIPT
- 10 RANGE
- 11 INVOKE
- 12 STACK OVERFLOW
- 13 REDIMENSIONED ARRAY
- 14 DIVISION BY ZERO
- 15 ILLEGAL DIRECT
- 16 TYPE MISMATCH
- 17 STRING TOO LONG
- 18 FORMULA TOO COMPLEX
- 19 CAN'T CONTINUE
- 20 UNDEFINED FUNCTION
- 22 SOS CALL
- 23 FILES BUSY
- 24 NOT SOS
- 25 I/O
- 26 FILE TOO LARGE
- 27 WRITE PROTECT
- 29 BAD PATH
- 30 FILE NOT FOUND
- 31 PATH NOT FOUND
- 32 VOLUME NOT FOUND
- 33 DUPLICATE FILE
- 34 DISK FULL
- 35 FILE LOCKED
- 36 FILE NOT OPEN
- 37 DEVICE NOT FOUND
- 253 EXTRA IGNORED
- 254 REENTER
- 255 BREAK(CONTROL C)

Business BASIC Reserved Words

C

The following is a list of the reserved words in Apple Business BASIC. Note that some must end with a left parenthesis to be considered reserved words. For example, AND is an illegal variable name, but ABS is not.

Material in this appendix is copyright 1983, Apple Computer, Inc., and used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA 95014.

ABS(DEF	GOTO
AND	DEL	HEX\$(
AS	DELETE	HOME
ASC(DIM	HPOS
ANT(DIV	IF
BUTTON(ELSE	IMAGE
CAT	END	INPUT
CATALOG	EOF	INSTR(
CHAIN	ERR	INT(
CHR\$(ERRLIN	INVERSE
CLEAR	EXFN	INVOKE
CLOSE	EXFN%	KBD
CONT	EXP(LEFT\$(
CONV(EXTENSION	LEN(
CONV\$(FN	LET
CONV%(FOR	LIST
CONV&(FRE	LOAD
COS(GET	LOCK
DATA	GOSUB	LOG

MID\$(REC(STR\$(
MOD	REM	SUB\$(
NEW	RENAME	SWAP
NEXT	RESTORE	TAB(
NORMAL	RESUME	TAN(
NOT	RETURN	TEN(
NOTRACE	RIGHT\$(TEXT
OFF	RND(THEN
ON	RUN	TO
OPEN	SAVE	TRACE
OR	SCALE(TYP(
OUTPUT	SGN(UNLOCK
PDL(SIN(USING
PERFORM	SPC(VAL(
POP	SQR(VPOS
PREFIX\$	STEP	WINDOW
PRINT	STOP	WRITE
READ		

System Error Messages **D**

System errors may occur when you first start up the Apple III or when you use the System Configuration Program or the Filer. Material in this appendix is copyright 1983, Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA, 95014.

ERRORS DURING STARTUP

Errors that may occur during the startup process are divided into two groups: diagnostic messages and SOS errors.

Diagnostic Messages

When you turn on your Apple III, it performs a number of diagnostic checks on its hardware. If any of these tests fails, you will get one of the following diagnostic messages. If you get one of these messages, see your dealer. Do not use the computer: a defective computer could erase data from your disk.

RAM The test of the Apple III's random-access memory failed. The pattern on the screen indicates which component is faulty. See your dealer.

ROM The test of the Apple III's read-only memory failed. See your dealer.

ACIA The test of the Asynchronous Communications Interface Adapter (ACIA), which controls the RS-232-C serial port, failed. See your dealer.

VIA The test of the Versatile Interface Adapters (VIAs), which control various internal functions in the Apple III, failed. See your dealer.

A/D The test of the Analog-to-Digital converter, which controls the joysticks, failed. See your dealer.

ZP The Zero Page test, which determines whether the Apple III's memory addressing circuitry is operational, failed. See your dealer.

SOS Errors

If the Apple III passes the diagnostic test, it tries to boot the disk in the built-in drive. If an error occurs during the boot process, a message is displayed (in black characters on a white background) in the middle of the video screen, the Apple beeps, and the system waits for you to try again. Any boot error is a fatal error: you must insert a proper boot disk, hold down the **CONTROL** key, and press the **RESET** button to reboot, generally with a different disk.

The following errors can be produced during a bootstrap operation:

(Blank screen) The disk in the built-in disk drive is not a boot disk. Insert a proper boot disk into the built-in disk drive, hold down the **CONTROL** key, and press **RESET** to attempt to reboot.

RETRY The disk boot failed. Either there is no disk in the built-in disk drive; or the disk is unformatted; or the data on the disk has been destroyed; or the disk is not inserted straight. Remove and reinsert the disk, making sure it is straight. If the error occurs repeatedly, place a known good boot disk in the built-in drive, close the door, and attempt to reboot. If the error occurs with several boot disks, see your dealer.

DRIVER FILE NOT FOUND There is no file name **SOS.DRIVER** listed in the volume directory of the boot disk. **SOS** cannot operate without device drivers, and the drivers must be stored in a file with this name in the volume directory of the disk.

DRIVER FILE TOO LARGE The **SOS.DRIVER** file is too large to fit into the system's memory along with the interpreter. Use the System Configuration Program to remove one or more drivers from this file.

EMPTY DRIVER FILE The **SOS.DRIVER** file contains no device drivers. **SOS** requires at least one device driver, usually **.CONSOLE**, to operate most interpreters.

INCOMPATIBLE INTERPRETER The interpreter is either too large or specifies a loading location that conflicts with **SOS**. This error

usually occurs when trying to load an older interpreter with a newer version of SOS.

INTERPRETER FILE NOT FOUND There is no file named SOS.INTERP listed in the volume directory of the boot disk. SOS cannot operate without an interpreter, and the interpreter must be stored in a file with this name in the volume directory of the disk.

INVALID INTERPRETER FILE The SOS.INTERP file is not in the proper format for an interpreter file. This error occurs when a file that is not an interpreter is assigned the name SOS.INTERP and the system is then booted.

INVALID DRIVER FILE The SOS.DRIVER file is not in the proper format for a driver file. Make sure that the file was either created by the System Configuration Program or obtained from a valid Apple III boot disk.

I/O ERROR The loader has encountered an I/O error while trying to read the kernel, interpreter, or driver file from the disk in the Apple III's internal disk drive. Make sure the disk is correct and properly inserted in the drive. If the error occurs with several boot disks, see your dealer.

KERNEL FILE NOT FOUND There is no file named SOS.KERNEL listed in the volume directory of the boot disk. The files SOS.KERNEL, SOS.INTERP, and SOS.DRIVER must all be present in the volume directory of a disk to be booted.

ROM ERROR: PLEASE NOTIFY YOUR DEALER Your Apple III contains an older version of the bootstrap ROM, which is not supported by this version of SOS. Your Apple dealer should be able to replace the ROM at no cost. If you receive this message, contact your dealer or the nearest Apple Service Center.

TOO MANY BLOCK DEVICES The SOS.DRIVER file contains too many block device drivers. Use the System Configuration Program to remove one or more of the block device drivers from this file.

SCP ERRORS

These errors are reported during the operation of the System Configuration Program.

No device drivers have been read You're not allowed to proceed until you've read at least one device driver.

Errors in Reading Driver Files

Illegal wildcard Either two wildcards were used where only one is allowed, or a wildcard was used when only a single file is allowed (for example, you can't send a directory listing to multiple files).

Invalid pathname The specified pathname violates the syntax rules for pathnames. A common error is more than 15 characters in a local name.

Pathname too long The specified pathname has more than 80 characters.

Errors in Editing System Parameters

File contents incorrect The character set or keyboard set contains invalid data.

Slot number may not be altered for this driver This driver works only if its device is in a particular slot.

Errors While Generating System

These errors will not prevent SCP from generating the system configuration, but they indicate that the system may not work.

Character set not loaded Use the Edit System Parameters option to set the required items.

Duplicate driver names Use the Edit Driver Parameters option to change the driver name.

Keyboard layout not loaded Use the Edit System Parameters option to set the required items.

One or more drivers require slot assignments Use the Edit System Parameters option to set the required items.

System parameters are not set Use the Edit System Parameters option to set the required items.

FILER ERRORS

The following errors are reported during the operation of the Filer:

Bad disk medium/drive The disk has been physically damaged and is unusable, or the disk drive is malfunctioning.

Blocked volume name expected You specified a character device name when a block device name was expected.

Cannot read source disk The disk that you are trying to make a copy of is unreadable. Make sure you have specified the proper source and destination drives and that the disk is correctly inserted in the source drive.

Command requires SOS-format disk This command (for example, Make a subdirectory) works only on SOS disks, not on UCSD-format disks.

Device dependent error #N See the manufacturer's documentation. The specified device is not connected to the system or is turned off.

Device not configured into the system The driver for the specified device is not in the system configuration.

Device not on line The specified device is not connected, not turned on, or (if a block device) has an open door or contains no volume or an unformatted volume.

Directory already exists The specified operation will create a directory with the same name as one that already exists. As this operation will delete the old directory, you will be asked to confirm your intention.

Directory full The specified operation would put more files into a directory than it can hold. One solution is to make a larger subdirectory and copy the files to it.

Disk drive not present/not configured You specified a disk drive that you either have not physically installed in your system or have not configured your Utilities disk to recognize. Make sure that your daisy-chain cables are connected securely, and use the System Configuration Program to configure your boot disk for the proper number of disk drives.

Error #N Internal program error. The number "N" is returned by the language interpreter. See the manual for the language.

File already exists SOS does not allow two files to use the same pathname. The specified operation will create a file with the pathname of a file that already exists, or it will change the pathname of one existing file to that of another. If this happens, you will be asked for confirmation, and if you give it, the old file of the pathname will be deleted so that the new file can use the pathname.

File expected This operation works only on files, and you specified something that was not a file.

File or blocked volume expected This operation works only on files or block devices, and you have specified a character device.

File or character device expected This operation works only on files or character devices, and you have specified something else, like a block device.

File open: access not permitted This error usually occurs when you have tried to delete a file that is open for access, such as a program that is executing. If you tried to delete SYSTEM.STARTUP from a boot disk while that file's program was running you would get this error.

Formatter device driver not present in system The formatter device driver for a drive must be configured into the system before a drive can be used for formatting or copying volumes. These drivers can be found in the SOS.DRIVER file on the Utilities disk.

Illegal disk format for this operation Either you tried to do something to an SOS disk that can only be done to a UCSD disk, or you tried to do something to a UCSD disk that can only be done to SOS disks.

Illegal character in volume name The volume name must contain only letters, numbers, and periods; the first character must be a letter.

Illegal device name The device name must start with a period, followed by a letter, followed by no more than 13 letters and numbers.

Illegal volume name length The volume name must not exceed 15 characters.

Illegal wildcard Either two wildcards were used where only one is allowed, or a wildcard was used when only a single file is allowed (for example, you can't send a directory listing to multiple files).

Invalid copy The specified copy operation (for example, copying a subdirectory to itself) is impossible.

Invalid number The number violates a syntax rule, usually because it contains a space or a comma.

Invalid pathname The specified pathname violates the syntax rules for pathnames. A common error is more than 15 characters in a local name.

I/O error during format operation Unidentified input/output error, such as an open drive door, no disk, or an unformatted disk.

I/O error Unidentified input/output error, such as an open drive door, no disk, or an unformatted disk.

Medium is write-protected or not present The disk you are trying to format or copy onto is write-protected or is not present. If you wish

to destroy the information on a write-protected disk, remove the disk from the drive, peel off the small tab that covers the write-protect notch, and reinsert the disk. If the disk is not write-protected, the drive may be configured but unplugged.

Not enough memory (too many files at once) The specified directory is too large to read into memory.

Not enough memory to do copy The volume directory has filled all available memory, leaving no room for storing information to be copied to another volume. The usual causes are too many device drivers configured into the system and too large a directory to fit into memory. The solution may be to remove some drivers from SOS.DRIVER. See the *Standard Device Drivers Manual* for details.

No room on volume The specified operation would try to put more information on a volume than it can hold.

Pathname too long The specified pathname has more than 80 characters.

Root directory expected This operation works only on a volume or root directory, and you specified something else (for example, Verify works only on volumes, not files).

Subdirectory expected This operation works only on subdirectories, and you have specified something else.

Subdirectory not found The specified subdirectory is not on the specified volume or device.

Unable to read file or directory A file or directory contains invalid data.

Volume already on line The specified volume has the same name as another volume already in the system. If you copy the Utilities diskette to a volume in another drive and then try to read the directory of the copy, you will get this error.

Volume not found The specified file was not in any of the devices known to the system, or the specified device has no volume in it.

Write-protect error The file or volume specified is write-protected and cannot be deleted or written to.

GENERAL ERRORS

SYSTEM FAILURE Indicates a catastrophic failure of SOS, from which the only recovery is to reboot your system. System failures are rare and can usually be attributed to sudden hardware failure or an

unknown error in the operating system or language interpreter. If you receive the same system failure at the same place in the same program more than once, your program has probably encountered an error in the language or operating system. Report such errors to your dealer, and fill out a User Input Report, so that they can be corrected in a future release of SOS.

- 01: Reentrant system call.
- 02: Interrupt not found.
- 03: Too many nested interrupts.
- 04: Unable to allocate NMI.
- 05: Event queue overflow.
- 06: Stack overflow.
- 07: Invalid request code.
- 08: Reserved.
- 09: Memory size less than 64K bytes.
- 0A: Invalid volume control block.
- 0B: Invalid file control block.
- 0C: Invalid allocation blocks.
- 0D: Pathname buffer overflow.
- 0F: Invalid buffer number.
- 10: Invalid buffer request.

Console Reference

E

Material in this appendix is copyright 1983 by Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA 95014.

KEYBOARD CODES

The ASCII codes and symbols for the keys of the Apple III (shown in Figure E-1) are listed in Table E-1. The modifier keys that do not generate ASCII codes operate as described in Table E-2. The special keys listed in Table E-3 generate the codes shown.



Figure E-1. Apple III keyboard

Table E-1. Standard Key Codes

Key Label	ASCII Codes and Symbols			
	Alone	SHIFT	CONTROL	Both
!	49 1	33 !	49 1	33 !
2 @	50 2	64 @	50 2	0 NUL
3 #	51 3	35 #	51 3	35 #
4 \$	52 4	36 \$	52 4	36 \$
5 %	53 5	37 %	53 5	37 %
6 ^	54 6	94 ^	54 6	30 RS
7 &	55 7	38 &	55 7	38 &
8 *	56 8	42 *	56 8	42 *
9 (57 9	40 (57 9	40 (
0)	48 0	41)	48 0	41)
- _	45 -	95 _	45 -	31 US
= +	61 =	43 +	61 =	43 +
\	92 \	124	28 FS	127 DEL
Q	113 q	81 Q	17 DC1	17 DC1
W	119 w	87 W	23 ETB	23 ETB
E	101 e	69 E	5 ENQ	5 ENQ
R	114 r	82 R	18 DC2	18 DC2
T	116 t	84 T	20 DC4	20 DC4
Y	121 y	89 Y	25 EM	25 EM
U	117 u	85 U	21 NAK	21 NAK
I	105 i	73 I	9 HT	9 HT
O	111 o	79 O	15 SI	15 SI
P	112 p	80 P	16 DLE	16 DLE
[{	91 [123 {	27 ESC	27 ESC
] }	93]	125 }	29 GS	29 GS
' ~	96 '	126 ~	96 '	126 ~
A	97 a	65 A	1 SOH	1 SOH
S	115 s	83 S	19 DC3	19 DC3
D	100 d	68 D	4 EOT	4 EOT
F	102 f	70 F	6 ACK	6 ACK
G	103 g	71 G	7 BEL	7 BEL
H	104 h	72 H	8 BS	8 BS
J	106 j	74 J	10 LF	10 LF
K	107 k	75 K	11 VT	11 VT
L	108 l	76 L	12 FF	12 FF
;	59 ;	58 :	59 ;	58 :
“ ”	39 ‘	34 “	39 ‘	34 “
Z	122 z	90 Z	26 SUB	26 SUB
X	120 x	88 X	24 CAN	24 CAN
C	99 c	67 C	3 ETX	3 ETX
V	118 v	86 V	22 SYN	22 SYN
B	98 b	66 B	2 STX	2 STX

Table E-1. Standard Key Codes (*continued*)

Key Label	ASCII Codes and Symbols			
	Alone	SHIFT	CONTROL	Both
N	110 n	78 N	14 SO	14 SO
M	109 m	77 M	13 CR	13 CR
, <	44 ,	60 <	44 ,	60 <
. >	46 .	62 >	46 .	62 >
/ ?	47 /	63 ?	47 /	63 ?

Table E-2. Modifier Keys

Key	Modifies
SHIFT	Standard keys, by keyboard layout table
CONTROL	Standard keys, by keyboard layout table
OPEN APPLE	Special and standard keys (sets high bit)
CLOSED APPLE	Special and standard keys (auto-repeat)
ALPHA LOCK	Standard keys, alphabetic only

Table E-3. Special Key Codes

Label	Code	Name	Label	Code	Name
ESCAPE	27	ESC	1	49	1
TAB	9	HT	2	50	2
RETURN	13	CR	3	51	3
ENTER	13	CR	4	52	4
spacebar	32	SPACE	5	53	5
LEFT ARROW	8	BS	6	54	6
RIGHT ARROW	21	NAK	7	55	7
UP ARROW	11	VT	8	56	8
DOWN ARROW	10	LF	9	57	9
- (minus)	45	-	0	0	0

CURSOR COMMAND KEYS

To enter cursor move mode, press **ESCAPE** once.

While the cursor move mode is active, the cursor appears as an inverse video plus sign. Table E-4 identifies the actions of the keys that may be used in cursor move mode.

To leave cursor command mode, press any key not listed.

CONSOLE CONTROL KEYS

A console control command is issued by holding down the **CONTROL** key while typing the appropriate digit on the numeric keypad. Typing the control sequence on the main keyboard will not work. The functions of the console control commands are listed in Table E-5.

Table E-4. Key Functions in Cursor Move Mode

Character	Function
LEFT ARROW or CONTROL H	Move cursor one space left
RIGHT ARROW or CONTROL U	Move cursor one space right
UP ARROW or CONTROL K	Move cursor one line up
DOWN ARROW or CONTROL J	Move cursor one line down
H or h	Home cursor
S or s	Clear entire viewport
P or p	Clear to end of viewport
L or l	Clear to end of line
V or v	Reset viewport to maximum size (full screen)
T or t	Set top-left corner of viewport
B or b	Set bottom-right corner of viewport

Table E-5. Console Control Commands

Keystroke	Function
CONTROL 5	Toggle Video Output. Turns the video display off and on. Program execution continues unaffected. If a program requests input from the console with the video display off, it is turned back on.
CONTROL 6	Flush type-ahead buffer. Any characters you have typed that have not yet been transferred out of the type-ahead buffer are discarded.
CONTROL 7	Suspend Screen Output. If you type this console command while the Apple III is sending output to the screen, output and program execution will stop. To continue, type the console command again.
CONTROL 8	Display Screen Control Characters. When you type this console command, the console driver will stop responding to control characters in the data sent to the display and will instead display their abbreviations. To restore normal operation, type the console command again.
CONTROL 9	Flush Output. All characters sent to the screen after you type this console command will be ignored. Program execution will continue, but no output will be displayed. To restore normal operation, type the console command again.

SCREEN CONTROL CODES

The screen control codes used to control the console screen are listed in Table E-6. You can use these codes with the CHR\$ function explained in Chapter 7. The arguments color, mode, and move are defined in Tables E-7, E-8, and E-9. The x, y, and shift arguments are single characters with values from 0 to 255. The argument shift is interpreted as two's complement.

Table E-6. Screen Control Codes

Code	Name	Args	Function
00	NUL		No operation
01	SOH		Save and reset viewport
02	STX		Set upper-left corner of viewport
03	ETX		Set lower-right corner of viewport
04	EOT		Restore viewport
05	ENQ		Turn cursor on
06	ACK		Turn cursor off
07	BEL		Sound the bell
08	BS		Move cursor left
09	HT		Move cursor right
10	LF		Move cursor down
11	VT		Move cursor up
12	FF		Home cursor
13	CR		Return cursor
14	SO		Turn screen off
15	SI		Turn screen on (enable text mode)
16	DLE	mode	Set text mode
17	DC1		Set normal text
18	DC2		Set inverse text
19	DC3	color	Set foreground color
20	DC4	color	Set background color
21	NAK	move	Cursor movement controls
22	SYN		Synchronize screen
23	ETB	shift	Horizontal shift
24	CAN	x	Horizontal position
25	EM	y	Vertical position
26	SUB	x y	Absolute position
27	ESC		No operation; reserved for future use
28	FS		Clear viewport
29	GS		Clear to end of viewport
30	RS		Clear line
31	US		Clear to end of line

Table E-7. Color Codes

Name	Number	Name	Number
Black	0	Brown	8
Magenta	1	Orange	9
Dark Blue	2	Gray	10
Lavender	3	Pink	11
Dark Green	4	Green	12
Gray	5	Yellow	13
Medium Blue	6	Aqua	14
Light Blue	7	White	15

On a black-and-white monitor, the colors appear as a gray scale: color 0 is black, color 15 is white, and the colors in between represent progressively lighter shades of gray.

Table E-8. Screen Modes

Mode	Operation
0	40-column black and white
1	40-column color (or gray scale)
2	80-column black and white

Table E-9. Cursor Movement Options

Value	Char	Scroll	Wrap	New Line	Advance
0	0	no	no	no	no
1	1	no	no	no	yes
2	2	no	no	yes	no
3	3	no	no	yes	yes
4	4	no	yes	no	no
5	5	no	yes	no	yes
6	6	no	yes	yes	no
7	7	no	yes	yes	yes
8	8	yes	no	no	no
9	9	yes	no	no	yes
10	:	yes	no	yes	no
11	;	yes	no	yes	yes
12	<	yes	yes	no	no
13	=	yes	yes	no	yes
14	>	yes	yes	yes	no
15	?	yes	yes	yes	yes

Graphics Reference

F

The following is a brief summary of the graphics procedures and functions. The graphics screen modes are defined in Table F-1, the color numbers are defined in Table F-2, and the transfer options are defined in Table F-3.

Table F-1. Graphics Screen Modes

Mode	Operation
0	280 by 192, black-and-white
1	280 by 192, limited color
2	560 by 192, black-and-white
3	140 by 192, full color
4	280 by 192, black-and-white, alternate screen
5	280 by 192, limited color, alternate screen
6	560 by 192, black-and-white, alternate screen
7	140 by 192, full color, alternate screen

Table F-2. Color Codes

Name	Number	Name	Number
Black	0	Brown	8
Magenta	1	Orange	9
Dark Blue	2	Gray	10
Lavender	3	Pink	11
Dark Green	4	Green	12
Gray	5	Yellow	13
Medium Blue	6	Aqua	14
Light Blue	7	White	15

On a black-and-white monitor, the colors appear as a gray scale: color 0 is black, color 15 is white, and the colors in between represent progressively lighter shades of gray.

Table F-3. Transfer Options

Option	Function
0	Draw Color ("Draw")
1	Draw Color OR Screen Color ("Add")
2	Draw Color XOR Screen Color ("Invert")
3	(NOT Draw Color) AND Screen Color ("Remove")
4	(NOT Draw Color)
5	(NOT Draw Color) OR Screen Color
6	(NOT Draw Color) XOR Screen Color
7	Draw Color AND Screen Color

GRAPHICS PROCEDURES AND FUNCTIONS

INITGRAFIX

Resets graphics defaults as follows:

Full screen viewport (x=0 to 279, y=0 to 191)

Cursor in the lower left-hand corner (x=0, y=0)

Normal color table

Normal transfer function

Format: PERFORM INITGRAFIX

GRAFIXMODE

Sets the graphics mode and selects the display buffer.

Format: PERFORM GRAFIXMODE(*%Mode,%Buffer*)

GRAFIXON

Displays the currently used graphics buffer on the video display.

Format: PERFORM GRAFIXON

PENCOLOR

Selects the color used for plotting.

Format: PERFORM PENCOLOR(*%Color*)

FILLCOLOR

Selects the erase color.

Format: PERFORM FILLCOLOR(*%Color*)

SETCTAB

Changes the color table.

Format: PERFORM SETCTAB(*%Sourcecolor,%Screencolor,
%Resultcolor*)

XFROPTION

Transfer option.

Format: PERFORM XFROPTION(*%Option*)

VIEWPORT

Sets the graphics area of the screen.

Format: PERFORM VIEWPORT(*%Left,%Right,%Bottom,%Top*)

MOVETO

Moves the cursor to the coordinates you specify.

Format: PERFORM MOVETO(*%X-coordinate,%Y-coordinate*)

MOVEREL

Moves the cursor relative to the current cursor position.

Format: PERFORM MOVEREL(*%nX-coordinate,%nY-coordinate*)

DOTAT

Draws a dot at the coordinate you specify.

Format: PERFORM DOTAT(*%X-coordinate,%Y-coordinate*)

DOTREL

Draws a dot relative to the cursor position.

Format: PERFORM DOTREL(*%nX-coordinate,%nY-coordinate*)

LINETO

Draws a line from the current cursor position to the coordinates you specify.

Format: PERFORM LINETO(*%X-coordinate,%Y-coordinate*)

LINEREL

Draws a line relative to the current cursor position.

Format: PERFORM DOTREL(*%nX-coordinate,%mY-coordinate*)

FILLPORT

Erases the viewport of any current graphics.

Format: PERFORM FILLPORT

XYCOLOR

A function that returns the color of the current cursor location.

Format: *var%=EXFN%.XYCOLOR*

XLOC

A function that returns the x-position of the cursor.

Format: *var%=EXFN%.XLOC*

YLOC

A function that returns the y-position of the cursor.

Format: *var%=EXFN%.YLOC*

NEWFONT

Loads a new character font.

Format: PERFORM NEWFONT(*@Font%(0,0),%Width,%Height*)

SYSFONT

Returns to normal type font.

Format: PERFORM SYSFONT

DRAWIMAGE

Displays a previously created bit image on the screen.

Format: PERFORM DRAWIMAGE(*@BLOCK%(0,0),%Rowbytes,%Skip X-coordinate,%Skip Y-coordinate,%bitswidth,%bitsheight*)

GSAVE

Saves a non-bit-image graphic to disk.

Format: PERFORM GSAVE. "*pathname*"

GLOAD

Loads a previously created non-bit-image graphic into memory.

Format: PERFORM GLOAD. "*pathname*"

RELEASE

Releases all graphics memory space.

Format: PERFORM RELEASE

Printer Reference



The following information can be used with the System Configuration Program to modify the printer configuration block for the .PRINTER driver. The bytes of the printer configuration block are defined in Table G-1. The values for speed and communications format are defined in Tables G-2 and G-3.

Material in this appendix is copyright 1983 by Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA 95014.

Table G-1. Printer Configuration Block

Byte	Meaning
00	Printer Speed
01	Communications Format
02	Carriage Return Delay
03	Line Feed Delay
04	Form Feed Delay

Table G-2. Printer Speed

Value	Speed
03	110 baud (Teletypewriter speed)
04	134.5 baud
06	300 baud (A common telecommunications speed)
07	600 baud
08	1200 baud (A common printer speed)
09	1800 baud
0A	2400 baud
0C	4800 baud
0E	9600 baud

Table G-3. Communications Format

Value	Format
22	7 bits, odd parity
26	7 bits, even parity
2A	7 bits, mark parity
2E	7 bits, space parity
00	8 bits, no parity
42	6 bits, odd parity
46	6 bits, even parity
4A	6 bits, mark parity
4E	6 bits, space parity

RS-232-C Reference

H

The following information can be used with the System Configuration Program to modify the configuration block of the .RS232 driver. The configuration block is defined in Table H-1. The values for Baud rates and data formats are defined in Tables H-2 and H-3.

The configuration block is normally set-up as shown in the right-hand column of Table H-1. You can change the handshake protocol using the values shown in Table H-4. The configuration blocks for some common RS232 devices are shown in Table H-5.

Material in this appendix is copyright 1983 by Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA 95014.

Table H-1. .RS232 Configuration Block

Parameter Name	Byte	Possible Values	Normal Mode Value*
Baud rate	00	(see Table H-2)	06 (300 baud)
Data format	01	(see Table H-3)	22 (7 bits, odd parity)
Carriage return delay	02	00-FF	00 (no delay)
Line feed delay	03	00-FF	00 (no delay)
Form feed delay	04	00-FF	00 (no delay)
Communications protocol	05	80, 40, 00	00 (no protocol)
Control character 1	06	00-7F	13 (DC3; "XOFF")
Control character 2	07	00-7F	11 (DC1; "XON")
Maximum buffer level	08	01-FF	DF (223 characters)
Minimum buffer level	09	00-FE	84 (132 characters)
Data block length	0A	01-FF	50 (80 characters)
Hardware handshake	0B	00, 80	00 (mode disabled)
*Default mode and values			

Table H-2. .RS232 Baud Rates

Value	Speed	
03	110 baud	(Teletypewriter speed)
04	134.5 baud	
06	300 baud	(A common telecommunications speed)
07	600 baud	
08	1200 baud	(A common printer speed)
09	1800 baud	
0A	2400 baud	
0C	4800 baud	
0E	9600 baud	

Table H-3. RS232 Data Formats

Value	Format
22	7 bits, odd parity (odd number of 1s)
26	7 bits, even parity (even number of 1s)
2A	7 bits, MARK parity (parity bit always 1)
2E	7 bits, SPACE parity (parity bit always 0)
00	8 bits, no parity
42	6 bits, odd parity (odd number of 1s)
46	6 bits, even parity (even number of 1s)
4A	6 bits, MARK parity (parity bit always 1)
4E	6 bits, SPACE parity (parity bit always 0)

Table H-4. .Values for Optional Protocols

Byte	XON/XOFF	ENQ/ACK	Hardware Handshake
05*	80 (XON/XOFF)	40 (ENQ/ACK)	00 (no protocol)
06	13 (DC3; XOFF)	05 (ENQ)**	(not used)
07	11 (DC1;XON)	06 (ACK)	(not used)
08	01-FF (DF)	(not used)	01-FF (DF)
09	00-FE (84)	(not used)	00-FE (84)
0A	(not used)	01-FF	(not used)
0B	00	00	80

*The values of bytes 00 to 04 do not depend on mode.
 **For ETX/ACK protocol; this is 03 (ETX).

Table H-5. Values for Some Common Devices

Device and characteristics (baud.bits.parity.protocol)	Device configuration block byte: 00 01 02 03 04 05 06 07 08 09 0A 0B
Default values (300.7.odd.no protocol)	06 22 00 00 00 00 13 11 DF 84 50 00
Another Apple III (9600.7.odd.hdwr handshake)	0E 22 00 00 00 00 xx xx xx xx xx 80
Apple II 300.7.SPACE.no protocol)	06 2E 00 00 00 00 xx xx xx xx xx 00
DEC LA120 terminal (1200.7.SPACE.XON XOFF)	08 2E 00 00 00 80 13 11 DF 84 xx 00
DEC VT100 terminal (9600.7.SPACE.XON XOFF)	0E 2E 00 00 00 80 13 11 DF 84 xx 00
SOROC IQ120 terminal (9600.7.MARK.no protocol)	0E 2A 00 00 00 00 xx xx xx xx xx 00
Qume Sprint 5 printer (1200.7.odd.hdwr handshake)	08 22 00 00 00 00 xx xx DF 84 xx 80
Qume Sprint 5 printer (1200.7.odd.ETX ACK)	08 22 00 00 00 40 03 06 xx xx 6E 00
HP 7225 Plotter (2400.7.SPACE. hdwr handshake)	0A 2E 00 00 00 00 xx xx DF 84 xx 80

Audio Reference

Generating a tone is accomplished by sending the six-character string shown in Figure I-1 to the audio driver (.AUDIO). The parameters mode and string are single characters whose ASCII values fall within the ranges shown in Table I-1. The parameters count and time are two-byte integers, with the low-order byte sent first. The actual value of a two-byte integer is obtained by adding the ASCII value of the low-order character to the ASCII value of the high-order character times 256. Table I-1 shows the acceptable ranges for the parameters count and time.

The relationship between the count parameter and the frequency of the resulting tone is given by the formula:

$$\text{count} = \frac{509000}{\text{freq}} \qquad \text{freq} = \frac{509000}{\text{count}}$$

The values of the count parameter for the notes of the tempered (12 tone) scale are listed in Table I-2. Middle C corresponds to the value 1946. Note that the lower the number, the higher the pitch.

The relationship between the time parameter and the duration of the resulting tone is given by the formula:

$$\text{duration} = \frac{\text{time}}{60}$$

Material in this appendix is copyright 1983, Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA, 95014.

Character:	1	2	3	4	5	6
Maximum Value:	128	63	255	63	44	1
Meaning:	Mode Volume		Count		Time	

Figure I-1. Audio parameters string

Table I-1. Range of Audio Parameters

Variable	Range
Mode	128
Volume	0 to 63
Count	100 to 16383
Time	0 to 300

Table I-2. Count Parameter Values for the 12 Tone Scale

Pitch	Lowest			Octave			Highest	
	1	2	3	4	5	6	7	8
B	8245	4122	2061	1031	515	258	129	
A#	8735	4368	2184	1092	546	273	136	
A	9255	4627	2314	1157	578	289	145	
G#	9805	4902	2451	1226	613	306	153	
G	10388	5194	2597	1298	649	325	162	
F#	11006	5503	2751	1376	688	344	172	
F	11660	5830	2915	1457	729	364	182	
E	12353	6177	3088	1544	772	386	193	
D#	13088	6544	3272	1636	818	409	204	102
D	13866	6933	3467	1733	867	433	217	108
C#	14691	7345	3673	1836	918	459	230	115
C	15564	7782	3891	1946	973	486	243	122

ASCII Character Codes **J**

The range of standard ASCII codes extends from 0 to 127. In Apple Business BASIC, the codes from 0 to 31 specify the control characters shown in Table J-1. Codes from 32 to 126 are used to specify the numbers, letters, and symbols shown in Table J-2. Apple Business BASIC also accepts codes from 127 to 255.

Material in this appendix is copyright 1983 by Apple Computer, Inc., and is used with permission of Apple Computer, Inc., 20525 Mariani, Cupertino, CA 95014.

Table J-1. ASCII Control Characters

Decimal	Hexa- decimal	Character	Keyboard Action	Comments and Notes
0	00	Null	CONTROL @	Null
1	01	SOH	CONTROL A	
2	02	STX	CONTROL B	
3	03	ETX	CONTROL C	Halts execution
4	04	ET	CONTROL D	
5	05	ENQ	CONTROL E	
6	06	ACK	CONTROL F	
7	07	BEL	CONTROL G	Beeps speaker
8	08	BS	CONTROL H	Backspace, (same as ←)
9	09	HT	CONTROL I	Horizontal tab
10	0A	LF	CONTROL J	Linefeed
11	0B	VT	CONTROL K	Vertical tab
12	0C	FF	CONTROL L	Formfeed
13	0D	CR	CONTROL M	Carriage return (same as RETURN)
14	0E	SO	CONTROL N	
15	0F	SI	CONTROL O	
16	10	DLE	CONTROL P	
17	11	DC1	CONTROL Q	
18	12	DC2	CONTROL R	
19	13	DC3	CONTROL S	
20	14	DC4	CONTROL T	
21	15	NAK	CONTROL U	
22	16	SYN	CONTROL V	
23	17	ETB	CONTROL W	
24	18	CAN	CONTROL X	Cancels line being edited
25	19	EM	CONTROL Y	
26	1A	SUB	CONTROL Z	
27	1B	ESC	ESCAPE	Cursor control and editing
28	1C	FS	CONTROL SLASH	
29	1D	GS	CONTROL RIGHT BRACKET	
30	1E	RS	CONTROL ^	
31	1F	US	CONTROL SHIFT UNDERLINE	

Table J-2. ASCII Letters, Numbers, and Symbols

Decimal	Hexadecimal	Character	Keyboard
32	20	Space	Spacebar
33	21	!	!
34	22	"	"
35	23	#	#
36	24	\$	\$
37	25	%	%
38	26	&	&
39	27	'	'
40	28	((
41	29))
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K

Table J-2. ASCII Letters, Numbers, and Symbols (*continued*)

Decimal	Hexadecimal	Character	Keyboard
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[[
92	5C	\	\
93	5D]]
94	5E	^	^
95	5F	_	_
96	60	,	,
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u

Table J-2. ASCII Letters, Numbers, and Symbols (*continued*)

Decimal	Hexadecimal	Character	Keyboard
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C		
125	7D	}	}
126	7E	~	~

Apple II Emulation

K

Packed with your Apple III is a disk marked "Apple II Emulation." When you use this disk, your Apple III will emulate an Apple II, allowing you to run most programs written for the Apple II. (We'll talk about the limitations shortly.)

When you boot from the Apple II Emulation disk, you turn your Apple III into the equivalent of an Apple II Plus with 48K of RAM, Applesoft II BASIC, a 16-sector Disk II controller card, and a serial card. With a few changes to the program, you can also emulate an Apple II with Integer BASIC, or either an Apple II or Apple II Plus with a communications card.

NOTE: When you use the Apple II Emulation disk, most of the advanced features of the Apple III won't be available to you. Also, you *can't* use the disk to emulate an Apple II with a Language Card, Applesoft card, or an Integer BASIC card.

Equivalent Slots

The Apple II Emulation disk sets up the Apple III's built-in disk drive as the equivalent of drive 1 (slot 6) on the Apple II. An additional Disk III is set up as the equivalent of drive 2 (slot 6).

In addition, the RS-232-C port of the Apple III will act like an Apple II serial card plugged into slot 7 or an Apple II communications card plugged into slot 5.

USING APPLE II EMULATION

Boot the Apple II Emulation disk just like any standard Apple III disk. After a few seconds, you'll see a menu that says

Apple II EMULATION MODE

RETURN – BOOT Apple II DISK

ESCAPE – CONFIGURATION MENU

You'll find that the majority of Apple II programs will run with no modifications needed. Thus, all you need to do is remove the Apple II Emulation disk from the built-in drive, insert the Apple II program you want to run, and press **RETURN**.

If the program doesn't run, reboot the Apple II Emulation disk and press **ESCAPE** to get to the configuration menu.

Apple II Emulation Options

Once you load the Apple II Emulation disk and press **ESCAPE**, you'll see a screen with six options. To select an option, use the four arrow keys. (If you're not sure what you need, see the documentation with your Apple II software.)

The six options are as follows:

1. **LANGUAGE.** Choose Applesoft or Integer BASIC. Both emulate an Apple II with an Autostart ROM. Integer BASIC includes Programmer's Aid #1.
Remember that you can only have one language in memory at a time.
2. **CARD.** A serial card sends data in one direction (out). It's normally used for a printer. A communications card is used for two-way communications using a modem.
3. **BAUD RATE.** This is the rate at which the Apple sends and receives data. A rate of 1200 is usually standard for a printer. Use either 300 or 1200 for a modem, depending on the speed of the modem.
4. **LINE FEED.** If your printer automatically sends a line feed after a carriage return, disable the line feed. If the line feed needs to come from the computer, enable it to do so.

5. **LINE WIDTH.** Set the length of the line to be printed. Since most Apple II programs use a 40-column line, 40 is a good choice. A width of 80 is “standard.”
6. **CARRIAGE RETURN DELAY.** Set this to ON if your printer needs extra time to return the print head to the left margin before it starts to print.

Once you've selected the options, all you need to do is remove the Apple II Emulation disk, place the Apple II software in the built-in drive, and press RETURN. Remember that the next time you boot the emulation disk, everything will be returned to the default (normal) values. If you want to save the configuration you've set up, hold down the CLOSED APPLE key and press RETURN.

EMULATION LIMITATIONS

Since the Apple III doesn't work exactly like an Apple II, there are a few limitations to keep in mind.

Software

You can only have one version of BASIC (either Applesoft or Integer BASIC) in memory at one time. To change BASICs, you'll have to reboot the Apple II Emulation disk.

You can't run Apple II Pascal programs that require the Language card. You can, however, run programs that use the Pascal Run-Time System.

Apple II emulation uses Apple II DOS 3.3. If you have DOS 3.2 disks, update them using the Apple II's FID and MUFFIN programs. If your disks are copy-protected, you can still run DOS 3.2 disks in the Apple II emulation mode by using the DOS 3.3 BASICS disk after you boot the Apple II Emulation disk.

Peripherals

You can't use the Apple II Language System, Applesoft II Card, or Integer BASIC card in the Apple III.

If your Apple II program requires a serial card in a slot other than 7 or a communications card in a slot other than 5, it won't run on the Apple III in emulation mode.

Since the Apple III doesn't have a cassette interface, any BASIC commands that use the cassette interface won't work.

The Apple II emulator recognizes Apple III joysticks as Apple II game paddles. However, since the Apple III uses a modified form of the Apple II's paddle-reading routines, an Apple II program that uses custom routines won't work in Apple II emulation mode.

Video

The Apple III's RGB (red-green-blue) output doesn't generate color output when used in emulation mode. You can only get color using the standard video output.

When high-resolution graphics are displayed, dots on the left border of the screen will flicker.

Keyboard

When you are in Apple II emulation mode, certain symbols can't be generated by pressing the associated keys on the Apple III keyboard. The affected keys are

@ ^ & (*) : + = - " ' ,

In addition, a few keys on the Apple III keyboard produce characters that the Apple II can't produce. When they're pressed while you're using the emulator, you'll see different characters on the screen. Refer to the *Apple III Owners Guide* for a list of these keys.

FINAL THOUGHTS

Despite the long list of limitations, you'll find that most Apple II programs will run in emulation mode with few, if any, problems. If you are writing a program from scratch, the best bet is to use Business BASIC and write them specifically for the Apple III.

Index

A

ABS, 121, 188
Acoustic couplers, 17
AND, 105
Applications programs, 12
Applications software, 59
Argument list, 159
Arguments, 120
 passing, 159
Arithmetic operators, 103, 189
Array dimensions, 101
Array name, 100
Arrays, 100-02, 189-90
 bit, 181-82
 dimensions of, 101
 element of, 100, 189
 subscript of, 100, 189
ASC, 122, 190
ASCII character codes, 261-65
ASCII codes, 240
ASCII control characters, 262
ASCII letters, 263-65
ASCII numbers, 263-65
ASCII symbols, 240, 263-65
Assembly language, 158
Assignment statements, 107-08
ATN, 121, 190
Audio, 259-60
Audio parameters, 182-83
 count, 182
 mode, 182
 range of, 260
 time, 182-83
 volume, 182

Audio parameters string, 260

B

Background (in graphics), 172
Backups, 32-37
 how to make, 35-37
BASIC, Business, introduction to,
 77-93
Baud rates, 256
Bit arrays, 181-82
Blank disks, formatting, 37-38
Block devices, 40
Boolean expressions, 105-06
Boot disk, 61
Booting, 4
Booting the system, 30
Branching, 110-12
 computed, 118-19
 conditional, 110, 111-12
 unconditional, 110
Buffer, 29
 display, 167
 typeahead, 29
Business BASIC
 commands in, 85-91
 control keys for, 91-93
 deferred execution of, 80, 83-84
 error messages for, 82, 221-28
 immediate execution of, 80
 introduction to, 77-93
 PRINT, used with, 81-82
 statements in, 85-91
 syntax of, 78-79

BUTTON, 190

Bytes, 10

C

Cassettes, 15

CAT, 89, 138, 190

CATALOG, 41, 143, 190

CHAIN, 155, 191

Character devices, 40

Character set, 72-74

CHR\$, 122, 135-36, 191

CLEAR, 108, 191

CLOSE, 147, 191

CLOSE#, 147, 191

Color codes, 245, 247

Colors, 167-68

Commands, 77

Commas, 127

Comment field, 69

Communications format, printer,
254

Communications software, 17

Compilers, 12

Composite video, 7

Computed branching, 118-19

Conditional branching, 110, 111-12

Configuration block, 69-70
printer, 253

.RS232 driver, 255

Console, 27, 239-45

Console control keys, 242-43

Constants, 95

numeric, 95

string, 95

CONT, 91, 192

CONTROL C, 91-92

Control characters, ASCII, 262

Control keys, 91-93

console, 242-43

CONTROL 5, 92, 192, 243

CONTROL 6, 92, 192, 243

CONTROL 7, 93, 192, 243

CONTROL 8, 93, 192, 243

CONTROL 9, 93, 192, 243

Controllers, 3

CONV, 121, 192

CONV\$, 121, 193

CONV&, 121, 193

CONV%, 121, 193

Copying files, 51-52

Correcting errors, 28-29

COS, 121, 193

COUNT, 184

Count parameter values, 260

CREATE, 142-43, 193-94

Cursor, 3

Cursor command keys, 242

Cursor movement options, 245

Cursor move mode, 84-85

D

Data, 79

DATA, 108, 194

Data formats, 256

Default, 49

Deferred execution, 80, 83-84

DEF FN, 123, 194

DEL, 87-88, 194

DELETE, 145, 195

Deleting files, 52-53, 144-45

Devices, 35-36, 40

listing configured, 57

Device drivers, 13, 30, 57, 60-62

Device handling utilities, 56-58

configured devices, listing, 57

copying volumes, 56

date, setting, 57-58

formatting disks, 56

renaming volumes, 56

time, setting, 57-58

verifying volumes, 57

Device name, 34, 67-68

Device subtype, 68

Device type, 68

Diagnostic messages, 231-32

DIM, 102, 195

Directories, 41-42

Disk

boot, 61

floppy, 4, 14-15

formatting blank, 37-38, 56

hard, 1, 15

how to care for, 30-32

system, 29-30

Disk care, 30-32

Disk drives, 4

Disk III, 14, 21

Disk III drives, 72

Display buffers, 167

DOTAT, 169, 174, 249

DOTREL, 169, 174, 249

DRAWIMAGE, 169, 181, 250

Driver status, 68-69

active, 68-69

inactive, 68-69

Duofile, 14, 15

E

Editing programs, 84
 ELSE, 195
 Emulation (of Apple II), 267-70
 keyboard options, 270
 limitations, 268
 options, 269-70
 peripherals, limitations, 268
 software, limitations, 268
 video options, 270
 END, 91, 117, 195
 Engineering notation, 134
 EOF, 196
 ERR, 196
 Error codes, 227-28
 Error messages, 82
 in Business BASIC, 221-28
 system, 231-38
 Errors
 correcting, 28-29
 Filer, 234-37
 general, 237-38
 SCP, 233-34
 SOS, 232-33
 ESCAPE, 27
 EXEC, 156-57
 EXFN, 160, 196
 EXFN%, 160, 196
 EXP, 121, 196
 Expansion slots, 3
 Exponent, 97
 Expressions, 103-06
 Boolean, 105-06
 nesting of, 104
 precedence in, 103-04
 relational, 104-05
 External routine, 158
 External subroutine, 159

F

Field, 46, 148
 comment, 69
 tab, 127
 File, 140-55
 how to copy, 51-52
 how to delete, 52-53, 144-45
 how to rename, 53-54, 145
 permanent, 39
 protecting, 145-46
 random-access, 148-49
 sequential-access, 147-48
 temporary, 40
 types of, 144

Filename, 40-41, 141
 Filer, 42, 48-49, 51-56
 copying files with, 51-52
 deleting files with, 52-53
 error messages for, 234-37
 renaming files with, 53-54
 set prefix option, 55-56
 subdirectories, creating, 54
 write-protection option, 55
 Fillcolor, 168
 FILLCOLOR, 169, 173, 248
 FILLPORT, 169, 250
 Floppy disks, 4, 14-15
 FOR, 113
 FOR and NEXT statements, 113-15
 Foreground (in graphics), 172
 Format specification, 128
 Formatting, 125
 Formatting blank disks, 37-38, 56
 FOR-NEXT loop, 114-15, 197
 FRE, 88, 197
 Function, 119-23, 159
 numeric, 120
 string, 120-22

G

General errors, 237-38
 Generating the system, 74-75
 GET, 110, 197-98
 GLOAD, 170, 175-76, 251
 GOSUB, 198
 GOTO, 110-11, 198
 Graphics, 165-81, 247-51
 Graphics mode, 8, 166
 Graphics routines, 169-70
 Graphics screen modes, 247
 GRAFIX, 170-72
 GRAFIXMODE, 169, 171, 248
 GRAFIXON, 169, 171, 248
 GSAVE, 169, 175, 251

H

Hard disk, 1, 15
 Profile, 15-16, 22-25
 HEX\$, 122, 198
 Hierarchical storage system, 42
 HOME, 136, 199
 HPOS, 139, 199

I

IF-GOTO, 111-12, 199
 IF-THEN, 111-12, 199
 IF-THEN-ELSE, 112

IMAGE, 130-31, 199-200
 Immediate execution, 80
 Immediate mode, 79
 INDENT, 136-37, 200
 Index variable, 113
 Infinite loop, 111
 INITGRAFIX, 169, 170-71, 248
 INPUT, 109-10, 200
 INPUT#, 151-53, 200
 Input/output, 125
 INSTR, 122, 200-01
 Instructions, 79
 INT, 121, 201
 Integer, 96, 98, 201
 Integer variables, 96
 Interface, 6
 RS-232-C serial, 6
 Internal subroutine, 159
 Interpreter, 12, 77
 INVERSE, 139-40, 201
 INVOKE, 158-59, 201-02

J

Joysticks, 5

K

KBD, 202
 Keyboard, 3-4, 27-29
 Keyboard codes, 239-41
 Key codes, 240
 special, 241
 Keypad, numeric, 92

L

Languages, 12-13
 LEFT\$, 122, 202
 LEN, 122, 202
 LET, 107, 202
 Line number, 79, 85-86
 LINEREL, 169, 175, 250
 LINETO, 169, 174-75, 249
 LIST, 86-87, 203
 Literal, 82
 LOAD, 90, 203
 LOCK, 145, 203
 LOG, 121, 203-04
 Logical operators, 105
 Long integer, 96, 98-99, 204
 Long integer variables, 96
 Loop
 infinite, 111
 nested, 114-15

M

Machine language, 12
 Manufacturer ID, 71
 Memory, 10-12
 random-access, 3, 10
 read-only, 11
 MID\$, 122, 204
 Mode, 7
 cursor move, 84-85
 graphics, 8, 166
 immediate, 79
 screen, 245
 text, 8
 Modem, 17, 26
 Modifier keys, 241
 Modulator, rf, 6
 Modulo, 103
 Monitor III, 6-7
 MOVEREL, 169, 174, 249
 MOVETO, 169, 173-74, 249
 Music, 165, 182-85

N

Nested loops, 114-15
 Nested subroutines, 117
 Nesting, 104
 NEW, 87, 88, 108, 204
 NEWFONT, 169, 181, 250
 NEXT, 113
 NORMAL, 139-40, 204
 NOT, 105
 NOTRACE, 161, 204
 Null line, 88
 Numeric functions, 120, 121
 Numeric keypad, 92

O

OFF EOF#, 155, 205
 OFF ERR, 161-62, 205, 221
 OFF KBD, 119, 206
 ON EOF#, 154-55, 205
 ON ERR, 161-62, 205, 221
 ON-GOSUB, 118-19, 206
 ON-GOTO, 118-19, 206
 ON KBD, 119, 206
 ON KBD GOTO, 119
 OPEN#, 146-47, 207
 Operands, 103, 189
 Operating systems, 13
 Optimizing programs, 162-64
 saving space, 162-63
 OR, 105
 Origin, 166

OUTPUT#, 154, 207
 OUTREC, 137, 207

P

Parameters, 67
 audio, 182-83
 system, 71-74
 Parameter values, count, 260
 Passing arguments, 159
 Path, 42
 Pathname, 42, 141
 PDL, 207-08
 Pencolor, 168
 PENCOLOR, 169, 172-73, 248
 PERFORM, 159, 208
 Peripherals, 17-18, 20, 22
 Permanent file, 39
 POP, 208
 Prefix, 55
 PREFIX\$, 141, 209
 PRINT, 81-82, 209
 PRINT#, 150, 209-10
 Printer, 16, 25-26
 Silentype, 5
 speed of, 254
 Printer configuration block, 253
 Printer reference, 253-54
 Printer speed, 254
 PRINT USING, 128-29, 209
 PRINT# USING, 150, 210
 Procedure, 159
 Profile hard disk, 15-16, 22-25
 Program, 83-88
 applications, 12
 deferred execution of, 83-84
 definition of, 83
 editing of, 84
 optimizing of, 162-64
 System Configuration, 13, 44,
 62-63, 65
 Prompt, 46
 Protecting files, 145-46
 Protocols, 256

R

RAM. *See* Random-access memory
 Random-access files, 148-49
 Random-access memory (RAM), 3,
 10
 READ, 108, 210
 READ#, 153-54, 210
 Read-only memory (ROM), 11
 Real numbers, 96, 97-98, 211

Real variables, 96
 Rear panel, 4-6
 REC, 211
 Records, 143
 Relational expressions, 104-05
 RELEASE, 170, 176, 251
 REM, 85-86, 211
 RENAME, 145, 211
 Renaming files, 53-54, 145
 Reserved variable, 136
 Reserved words, 99-100, 229-30
 RESET key, 4
 used like CONTROL C, 92
 RESTORE, 109, 211
 RESUME, 212
 RETURN, 212
 Rf modulator, 6
 RGB, 1, 7
 RIGHT\$, 122, 212
 RND, 121, 213
 ROM. *See* Read-only memory
 RS-232-C, 255-57
 RS-232-C serial interface, 6
 .RS232 configuration block, 255
 RUN, 90-91, 213

S

SAVE, 89, 213
 Saving space in programs, 162-63
 SCALE, 134, 213-14
 Scientific notation, 96, 133
 SCP. *See* System Configuration
 Program
 Screen control codes, 243-45
 Screen modes, 245
 Semicolons, 126-27
 Sequential-access files, 147-48
 Serial interface, RS-232-C, 6
 SETCTAB, 169, 177-78, 249
 Set prefix, 55-56
 SGN, 121, 214
 Silentype Printer, 5
 SIN, 121, 214
 Slot assignments, 72
 Slot number, 71
 Software, 12-13
 applications, 59
 communications, 17
 system, 59, 88
 Sophisticated Operating System
 (SOS), 13, 29, 59
 device manager of, 60
 error messages of, 232-33

SOS (*continued*)
 file manager of, 60
 interrupt manager of, 60
 memory manager of, 60
 SOS.DRIVER, 60
 SOS.KERNEL, 60
 utility manager of, 60
 SOS. See Sophisticated Operating System
 Sound, 165, 182-85
 SPC, 128, 214
 SQR, 121, 215
 Statements, 77
 STEP, 113-14, 215
 STOP, 91, 215
 Storage system, hierarchical, 42
 STR\$, 122, 215
 String, 96, 99
 audio parameters, 260
 String constant, 95
 String functions, 120-22
 String variables, 96
 SUB\$, 122, 215-16
 Subdirectories, 42
 how to create, 54
 Subroutine, 116-17
 external, 159
 internal, 159
 nested, 117
 SWAP, 107-08, 216
 Syntax, 78-79
 SYSFONT, 169, 181, 250
 System Configuration Program (SCP), 13, 44, 62-63, 65
 error messages, 233-34
 System disks, 29-30
 System error messages, 231-38
 System parameters, 71-74
 character set, 72-74
 Disk III drives, number of, 72
 slot assignments, 72
 System software, 59, 88
 System unit, 1, 9

T

TAB, 127, 216
 Tab fields, 127
 TAN, 121, 216
 Template, 128
 Temporary file, 40
 TEN, 122, 217
 TEXT, 139, 171-72, 217
 Text modes, 8

TRACE, 115, 161, 217
 Transfer option, 168, 248
 Truth tables, 106
 TYP, 217
 Typeahead buffer, 29

U

Unconditional branching, 110
 Unifile, 14, 15
 Unit number, 71
 Universal Parallel Interface Card, 25-26
 UNLOCK, 145, 218
 Unpacking, 19
 Utilities, 44, 46-48
 device handling, 56-58

V

VAL, 122, 218
 Variable, 95-97
 index, 113
 integer, 96
 long integer, 96
 real, 96
 reserved, 136
 string, 96
 types of, 96-97
 Variable name, 95-96
 Version ID, 71
 Video, composite, 7
 Video display, 6-9
 Viewport, 168
 VIEWPORT, 168, 169, 172, 249
 Volume name, 34, 43, 49
 VPOS, 139, 218

W

Warm boot, 4
 Wildcards, 47-48
 WINDOW, 138-39, 218-19
 Word wrap, 28
 WRITE#, 151, 219
 Write-protection, 33-34, 55

X

XFROPTION, 169, 178-80, 249
 XLOC, 169, 250
 XYCOLOR, 169, 180, 250
 X-y coordinate system, 166

Y

YLOC, 169, 250

The Osborne/McGraw-Hill Guide to Your Apple® III

All the information you need for trouble-free set-up and operation of your Apple® III and its peripherals — plus a section focusing on Business BASIC programming — is included in this easy-to-understand reference guide.

The Osborne/McGraw-Hill Guide to Your Apple® III is the single best source to show you how to:

- Use your microcomputer, printer, Profile hard disks, floppy disk drives and modem
- Develop practical programs with Business BASIC
- Create graphics and utilize the sound capabilities of your system
- Understand and install the Apple III SOS operating system.

Even if you're a beginning computer user, **The Osborne/McGraw-Hill Guide to Your Apple® III**, with its quick reference guide to Business BASIC and its comprehensive index and appendix for ease of use, will help you get your Apple III up and running in no time at all.

Apple is a registered trademark of Apple Computer, Inc.

ISBN 0-88134-101-0

