User's Manual for Single Floppy Disk Drive - Model 2031

commodore COMPUTER

Table of Contents

Chapter1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

List of Illustrations

List of Tables

Chapter 1

GENERAL DESCRIPTION

INTRODUCTION

With the purchase of your Commodore 2031 Floppy Disk Drive, you have greatly enhanced the computing power of your disk-oriented Commodore Computer system. The system has been designed around the central concept of providing you, the user, with large file-handling capabilities supported by BASIC programming commands and further simplified by Disk Operating System (DOS) Support command abbreviations.

Because the disk drive is an "intelligent" peripheral device, its operation requires no additional space in the computer's memory. This means you have just as much computer memory available to you as when the disk drive is not attached.

When the disk drive is properly interface to a Commodore Computer, YOU become an important element of the system. Your importance, however, is measured directly with how well you understand how to effectively utilize the hardware and software. That's why it is best to think of your computer and disk drive as only part of a system. Grasping and understanding the concept that each individual component acts and reacts to signals and commands from other devices in the system will accelerate your fundamental grasp of how to operate, control, and master the system. This concept forms the basis of this manual: teaching you how to gain mastery of the system so that it will perform as desired. That is the reason this manual is organized in a building block fashion; it permits you to advance as fast as you desire.

To get the most out of your system you should study your Computer User's Guide and, if necessary, obtain selected manuals listed in Table 1. You will benefit most if you first read through this entire manual before attempting to operate the Disk Drive.

THE MANUAL - A LEARNING TOOL

This manual is designed to encourage the concept of learning by doing. Difficult concepts and procedures have been broken down into simpler steps that lead the user through examples which provide ample opportunity to experiment later by returning to each command description. When possible, the command format has been included with the command description and, as an additional aid, a User's Quick Reference has been placed in Chapter 8 for easy access if problems persist.

The first chapters discuss basic hardware features and permit you, right from the start, to become familiar with the disk drive by doing the performance tests which comprise the first essential "hands on" experience. Then, by actually using your particular disk drive and learning to carefully follow instructions, you can gain the necessary confidence to proceed to more comprehensive subjects.

By the time you begin Chapter 3, Learning How To Use Your Floppy Disk Drive, you will have already used some portions of the DOS Support system which is fully described in Chapter 7. It is actually easier to instruct a new user by using the simplified command structure of DOS Support than attempting to explain the entire command hierarchy at an early stage. This concept of providing you with enough essential information to complete a task, a step-by-step description of the task, and meaningful examples is a feature of this manual which will provide you with sufficient incentive to actually complete the task.

Problems are often accompanied with error messages. These error messages are contained in Chapter 8 where they can be quickly referenced, if needed. The Error Message discussions have been expanded to include:

* How to request error messages.
* Error message summary.
* Detailed error message descriptions.

For those users who have been reluctant to attempt disk programming because of the presumed degree of difficulty, note that the entire disk command hierarchy is structured in this manual from the least difficult to the more complex:

* Commands for file manipulation and maintenance
* Commands for data handling
* Advanced programming
* Advanced file handling
* Simplified commands

Users who have attained some degree of programming skills may desire to begin with the advanced subjects such as random access or relative files while others may be content with just following the manual's format. In either case, this manual has been structured to provide the user with essential information in a logical sequence. Follow the examples, attempt the step-by-step procedures, and learn by doing.

GENERAL HARDWARE INFORMATION

The CBM model 2031 Floppy Disk Drive, described in this manual, is a 170 K-byte capacity storage device. Its primary components consist of read/write controls, drive motor electronics, one drive mechanism, one read/write head, and a track positioning mechanism. Also, the disk drive conforms to IEEE-488 interface requirements.

For ease of reference, the Model 2031 Single Floppy Disk Drive will be referred to in this manual as "the 2031".

The 2031 is operationally compatible with the following Commodore Computers:

1. Series 2001 -- 16K and 32K -- Operating With BASIC Version 3.0
2. Series 2001 -- PET 8K -- Upgraded to BASIC Version 3.0
3. Series 3000 -- 16K and 32K -- Operating With BASIC Version 3.0
4. Series 4000 -- PET 8K, 16K, and 32K -- Operating With BASIC Version 4.0
5. Series 8000 -- 32K -- Operating With BASIC Version 4.0

FRONT PANEL

The disk drive's front panel consists of an identification panel across the top, a slot in which to insert a diskette, and a door to close after inserting the diskette.

When the door is closed, the diskette is clamped onto the diskette spindle hub. Also, there is a LED indicator light on the front panel. This light is called the Drive Active Indicator; it lights when the drive is active. This LED flashes continuously if an error has occurred; it will extinguish after the error channel has been read.

Back Panel

The back of the disk drive contains an IEEE-488 interface connector. Near the panel's lower edge is the power ON/OFF switch. There is also a "slow blow" fuse, and the AC power cord.

Interior Configuration

The interior of your 2031 contains a disk drive and all the necessary logic for disk operation. The mechanical device is, for the most part, located beneath the disk spindle.

The Diskette

The diskette (also known as a minifloppy, floppy diskette, minidiskette, etc.) is similar to the standard flexible disk. You should ensure that you buy diskettes for SOFT SECTORED FORMAT. Your Commodore dealer can supply your needs.

Table I. SUGGESTED READING LIST

Pet/CBM Personal Computer Guide. C.S. Donahue and J.K. Enger, Osborne/McGraw-Hill. 630 Bancroft Way, Berkeley, CA 94710

Hands-On Basic With A Pet. H.D. Peckman, McGraw-Hill, 1979

Entering BASIC. J. Sack and J. Meadows, Science Research Associates, 1973

BASIC: A Computer Programming Language. C. Pegels, Holden-Day, Inc., 1973

BASIC Programming. J. Kemeny and T. Kurtz, Peoples Computer Co., 1010 Doyle (PO Box 3100), Menlo Park, CA 94025, 1967

A Guided Tour of Computer Programming In BASIC. T. Dwyer, Houghton Miflin Co., 1973

Programming Time Shared Computer In BASIC. Eugene H. Barnett, Wiley-Interscience, L C 72-175789

Table 1. Suggested Reading List (Continued)

Programming Language #2. Digital Equipment Corp., Maynard MA 01754

101 BASIC Computer Games. Software Distribution Center, Digital Equipment Corp., Maynard, MA 01754

What To Do After You Hit Return. Peoples Computer Co., 1010 Doyle (PO Box 3100), Menlo Park, CA 94025

Basic BASIC. James S. Coan. Hayden Book Co., Rochelle, Park NJ Workbooks 1-5 T.I.S., PO Box 921, Los Alamos, NM 87544

Programming The 6502. R. Zaks, Sybex, 1978

24 Tested, Ready-To-Run Game Programs In BASIC. K. Tracton, Tab Books, 1978

Some Basic Programs. M. Borchers and R. Poole. Osborne & Assoc. Inc., 1978

The Channel Data Book. B. Lewis, 5950 Mandarin Ave., Goleta, CA 93017, 1978

PET And The IEEE 488 Bus(GPIP). Osborne/McGraw-Hill, 630 Bancroft Way, Berkeley CA 94710

Specifications

Table 2 contains the specifications for the 2031.

CARE OF THE 2031

The disk drive should be placed on a flat surface free of vibration. It is important that dust particles be kept at a minimum since a particle buildup will interfere with optimum operation. If you should experience a hardware failure, contact your Commodore dealer. Any attempt to correct the problem yourself could result in voiding the warranty.

Table 2. SPECIFICATIONS: MODEL 2031 SINGLE DRIVE FLOPPY DISK

STORAGE	
Total capacity	174848 bytes
Sequential	168656 bytes
Relative	167132 bytes; 65536 records per file
Directory entries	144
Sectors per track	17 to 21
Bytes per sector	254
Tracks	35
Blocks	683
IC's:	
Controller and Interface	
6502	microprocessor
6522	I/O, interval timers
6522	I/O, interval timers
2114 (4)	4x1K RAM
PHYSICAL	
Material	18 ga. steel
Dimensions	
Height	6.5"
Width	15.0"
Depth	14.35"
ELECTRICAL	
Power requirements USA (domestic)	
Voltage	120 VAC
Frequency	60 Hertz
Power	50 Watts
Power requirements (international)	
Voltage	100, 220, or 240 VAC
Frequency	50 Hertz
Power	50 Watts
MEDIA:	
Diskettes	Standard mini 5 1/4", single sided, single density

CARE OF THE DISKETTES

Handle diskettes with care. Follow these instructions to maintain the quality of the diskette and to protect the integrity of the data:

1. Return the diskette to its storage envelope whenever it is removed from the drive.

2. Keep the diskettes away from magnetic fields. Exposure to a magnetic field can distort the data.

3. Never leave a diskette on top of your computer or disk drive.

4. Do not write on the plastic jacket with a lead pencil or ball-point pen. Use a felt tip pen or fill out the label before attaching it to the jacket.

5. Do not expose diskettes to heat or sunlight.

6. Do not touch or attempt to clean the diskette surface. Abrasions will cause loss of stored data.

7. Before applying power to the 2031, insure that there is no diskette in the drive.

UNPACKING THE DISK DRIVE

Before unpacking the disk drive, inspect the shipping carton for signs of external damage. If the carton is damaged, be especially careful when inspecting its contents. Carefully remove all packing material and the contents of the carton. DO NOT discard any packing material until you have made sure you have located all the contents of the carton! The carton should contain:

1. Commodore Single Floppy Disk Drive
2. User Manual, Number 2031036
3. A 2031 TEST/DEMO diskette

If any items are missing, please contact your Commodore dealer immediately.

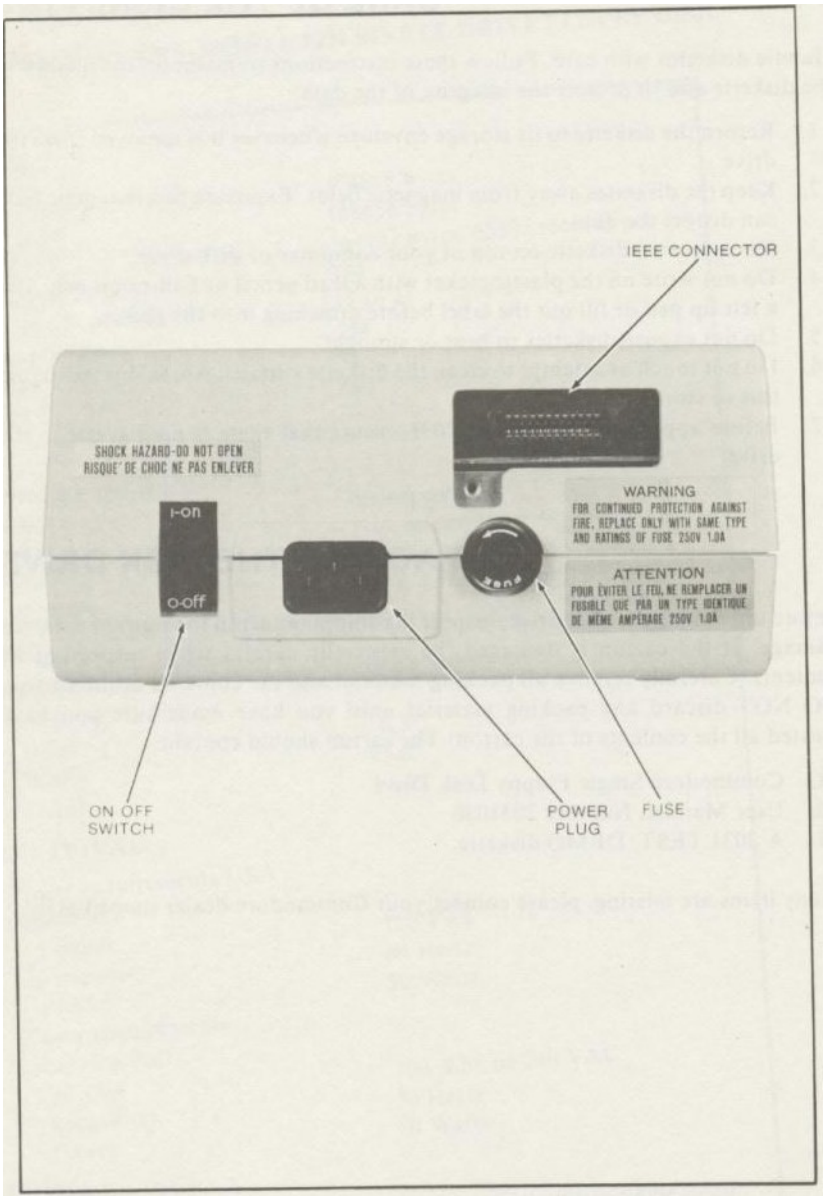

Figure 1. Model 2031 Rear View

CHAPTER 2

PREPARING TO USE YOUR DISK DRIVE

Before starting to use your disk drive, make sure it is in good working condition. This includes properly connecting it to your computer, giving it a power-on and initial checkout test, and finally the performance test using the appropriate TEST/DEMO diskette.

CONNECTING THE DISK DRIVE TO THE COMPUTER

One of two connector cables are required to interface the floppy to the computer. These cables can be supplied by your Commodore dealer.

1. PET-to-IEEE cable P.N 320101
   Use this cable if the disk drive is to be the only (or first) IEEE device connected to your computer.

2. IEEE-to-IEEE cable, P/N 905080
   Use this cable if your disk drive is to be connected ("daisy chained") to another peripheral device such as the Commodore Model 2022, or any other suitable interfaced printer.

Follow these steps to connect the disk drive to you computer:

STEP 1. Turn power OFF to the computer.

STEP 2. Place the disk in a convenient location as close as possible to the computer. DO NOT connect the disk drive to a power outlet at this time (see Figure 2).

STEP 3. Connect the PET-to-IEEE cable between the IEEE-488 interface connector on the computer and the connector on the disk drive. If additional IEEE devices are to be connected, the IEEE-to-IEEE cable(s) must be used.

STEP 4. Connect the disk drive power cable to an AC outlet. DO NOT turn on power at this time.

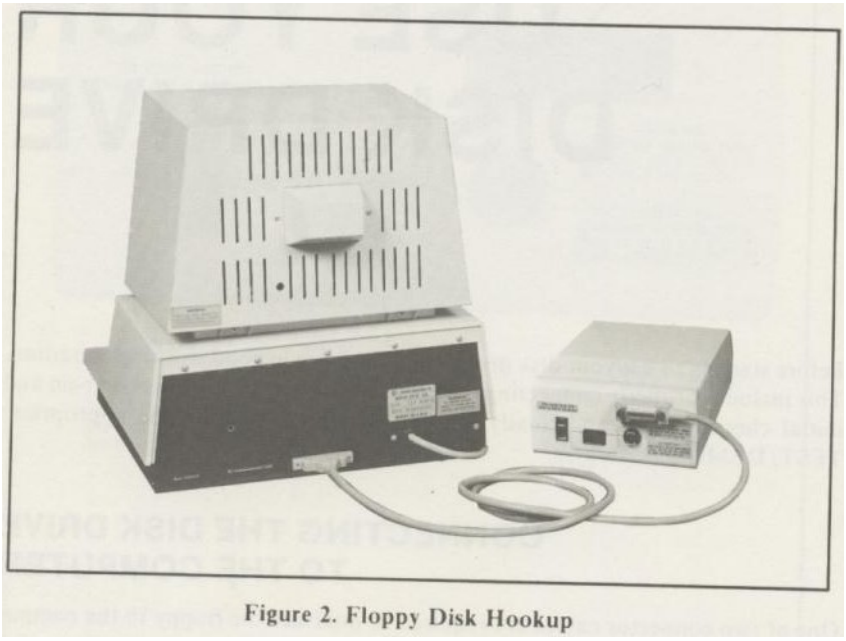

Figure 2. Floppy Disk Hookup

PERFORMING THE POWER-ON TEST

You are now ready to proceed with the power-on part of the checkout:

STEP 1. Open the disk drive door. Ensure that there is no diskette in the drive.

STEP 2. Turn power ON to the COMPUTER and verify that it is working properly.

STEP 3. Apply power to the disk drive. The LED flashes to indicate a faulty condition. The number of flashes determine which component is faulty. The following lists the components and their respective number of flashes.

Flash Code	Component
1	zero page, U3B, U3D
2	ROM U5F
3	ROM U5H
4	RAM U3B, U3C, U3D, U3E

NOTE

If the problem persists, try disconnecting the other devices attached to the IEEE bus. This should assure that a problem related to another device does not affect the disk drive.

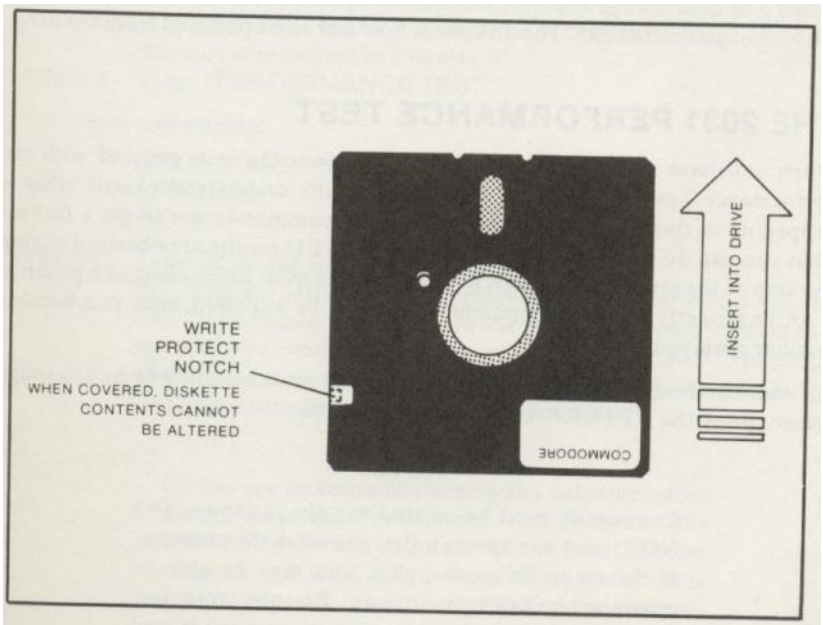

Figure 3. Position for Diskette Insertion

INSERTING THE DISKETTE INTO THE 2031

Caution

NEVER APPLY POWER TO THE DISK DRIVE IF A DISKETTE IS PRESENT (LOCKED AND SEATED).

STEP 1. Ensure that the power to the disk drive is OFF and DO NOT apply power until you complete this step. Open the disk drive door and ensure that no diskette is present in the drive.

STEP 2. If the preceding conditions have been met, you may apply power to the disk drive.

STEP 3. Insert the diskette into the slot with the write protect tab oriented to the left (see Figure 3) and push it completely in.

STEP 4. Once the diskette is in the slot, GENTLY close the door partially to center the diskette. Repeat this centering action several times and then close the door firmly. The diskette is now seated and locked in the drive, ready for processing by the computer.

STEP 5. To remove the diskette, insert your index finger under the lip of the door and gently PULL. This will release the door and permit access to the diskette. The diskette is now free to be removed from the drive.

THE 2031 PERFORMANCE TEST

When you have successfully completed the Power-On test, proceed with the Performance Test. Don't worry if you don't fully understand exactly what is happening in this test. At this point, enter the commands just to get a feel for what you can do with your disk. If UNEXPECTED result are obtained during any step of the test, stop and start over again. The most likely cause of a problem is an improperly entered command. This is to be expected until you become familiar with your disk nit.

All commands are entered via t5he keyboard and must be followed by a carriage return: press the RETURN key on your keyboard.

NOTE

Commands must be entered exactly as shown. DO NOT insert any spaces unless shown in the example. If the error indicator lights, you may be able to continue the example anyway. Re-enter your last command. If the light goes out, your correction was successful and you may continue.

NOTE FOR BUSINESS KEYBOARD USERS ONLY: You can set your computer for upper case character entry. Do this by typing:

POKE 59468,12 and pressing the RETURN key.

Although it is not absolutely necessary to give this command, it does permit easy entry. In addition, the examples in this manual can be duplicated exactly when you sue only upper case. If this step is omitted the display will be in lower case letters. DO NOT use the shift key when entering commands.

STEP 1. Insert the DEMO diskette into the drive as previously instructed.

STEP 2. Type: LOAD"*",8 and press RETURN. The computer will load the first file from the diskette in drive 0 and display the following:

```
READY.
LOAD"*",8
SEARCHING FOR *
LOADING

READY.
```

STEP 3. Type: RUN and press RETURN. This will cause the DOS Support Program to be executed. (This program is not necessary for the disk unit to operate; it just simplifies the direct mode commands.) A brief introductory description of DOS Support is presented in Chapter 3 as it applies to Chapter 3 operations. A detailed description of DOS Support is presented in Chapter 7.

STEP 4. Type: /PERFORMANCE TEST

The screen will display:

```
SEARCHING FOR PERFORMANCE TEST
LOADING
READY.
```

STEP 5: Type: RUN and press RETURN.
The program instructs you to place a blank diskette into the drive. The Performance Test Program executes a shortened version of the test used by Commodore in final inspection of the 2031. The purpose of this test is to ensure that the unit is functioning correctly and will take approximately seven minutes to complete.

NOTE

Do not use diskettes containing any valuable information since the Performance Test Program will re-format them and any data will be lost. The test program will label this diskette "Test Disk." This diskette is ready for further use when the test program is completed and the performance test has been satisfied.

The computer will first format the diskette. This procedure takes up to three minutes. At the end of the operation the screen displays:

```
NEW-0 COMMAND OK 0 00.0

THE DRIVE PASSES THE MECHANICAL TEST
```

The computer conducts the remainder of the Performance Test and displays:

```
OPEN WRITE FILE ON 0          OK 0  00.00
WRITING DATA TO 0             OK 0  00.00
CLOSE WRITE DATA TO 0         OK 0  00.00
OPEN READ FILE ON 0           OK 0  00.00
READING DATA FROM 0           OK 0  00.00
SCRATCH FILE ON 0
   FILES SCRATCHED 1          01.00
WRITE TRACK XX ON 0           OK 0  00.00
WRITE TRACK   1 ON 0          OK 0  00.00
READ TRACK XX ON 0            OK 0  00.00
READ TRACK    1 ON 0          OK 0  00.00

    UNIT HAS PASSED PERFORMANCE TEST!

REMOVE DISKETTE FROM THE DRIVE BEFORE TURNING
POWER OFF.

READY.
```

STEP 6. Remove the diskettes and return them to their protective jackets. The floppy has passed the Performance Test.

STEP 7. If any problems have been encountered during this phase of the test, return to step 1 and repeat the entire procedure. If problems persist and you do not reach a satisfactory conclusion to the Performance Test, contact your Commodore dealer.

Chapter 3

LEARNING HOW TO USE YOUR FLOPPY DISK DRIVE

Your CBM Floppy Disk Drive, which enhances your computing power with added storage and file handling capability, is controlled directly with:

* BASIC commands entered via the keyboard
* BASIC statements within programs
* Special disk commands

In this chapter you will learn how to apply those commands and statements. This chapter is organized in such a way that the functions and format of disk commands are described in a manner which permits the user to perform disk-related tasks. For BASIC 4.0 users, those BASIC commands which correspond to each disk maintenance command are also discussed.

Before using your floppy disk make sure you know how to:

As changes occur to the BAM in DOS memory, the BAM on disk will be updated to reflect these changes. Updates to the BAM occur when a program is SAVEd or a CLOSE or DCLOSE is performed on a new RELative or SEQuential data file.

1. Operate you Commodore Computer
2. Do elementary programming in BASIC
3. Open and close files.

NOTE

> The BASIC statements described in this chapter apply specifically to the 2031. Certain commands and statements may follow a slightly different format or produce different results from those described herein when they are used with the computer or with other peripherals. Consult the appropriate manual for the exact use of these commands and statements in other applications.

This chapter will first acquaint the user with those fundamental disk commands that perform disk maintenance and file manipulation and will then progressively advance toward an understanding of those BASIC commands used for data handling. Approached in this manner, the user will then have developed the necessary confidence and programming skills to proceed to advanced disk programming techniques. Practice the disk commands, read the examples, and follow the step-by-step illustrations of their usage. The understanding of the more advanced disk programming techniques will depend to a large degree upon how well the fundamentals have been mastered.

To facilitate your understanding and mastery of Commodore BASIC, two computer terms are stressed in this Chapter: Block Availability Map (BAM) and Disk Operating System (DOS). Although these are conventional terms, they will be briefly discussed as they relate to the Commodore Floppy Disk Drive.

THE BLOCK AVAILABILITY MAP (BAM)

The BAM is a disk memory representation of available and allocated space on a disk. When the system stores information on a disk, the BAM will be automatically referenced by the DOS to determine what space is available and how many blocks can be allocated. If sufficient space is available to store a given file, it will be stored on the disk and the BAM updated to account for the space allocated. However, if the DOS detects that a file will occupy more space than available, an error message will be generated.

Formatting a disk creates the BAM which is then loaded into DOS memory upon initialization. The BAM which is then loaded into DOS memory upon initialization. The BAM is stored on diskette on track 18, sector 0, and 128 bytes in length.

As changes occur to the BAM in DOS memory, the BAM on disk will be updated to reflect these changes. Updated to the BAM occur when a program is SAVEd or a CLOSE or DCLOSE is performed on a new RELative or SEQuential data file.

THE DISK OPERATING SYSTEM (DOS)

The DOS is responsible for managing information exchange between disk controller and the computer.

The DOS performs many functions which are transparent to the user but which are vital to the operation of the system. For example, the DOS monitors the input/output (I/O) of the disk so that channels are properly assigned and that no lengthy waits for an open channel occur. In addition to monitoring of disk I/O, the DOS also uses the channel structure to search the directory and to delete and copy files.

There is another function of DOS called DOS Support which was used during hardware checkout in Chapter 2. Review the Performance Test procedure and observe the special symbols of DOS Support which were used to duplicate and initialize the disks before these procedures were fully explained to the user. Because of its ease of use, DOS Support symbols were easier to implement at that point than attempting to explain the programming procedures they replace. It is now appropriate to briefly discuss how DOS Support can enhance and simplify you knowledge of operating your Commodore computer.

The first file on the TEST DEMO diskette that comes with your disk drive is the DOS SUPPORT PROGRAM. This program, when loaded into computer memory, permits the user to enter abbreviations from many disk commands.

For example, disk commands which would normally be transmitted to the disk using the PRINT#1fn,”commandstring” format may be transmitted via DOS SUPPORT by preceding the command with > or @. Typing slash (/) followed by a program name and RETURN will cause DOS SUPPORT to load that program into memory. Replacing certain disk commands with DOS Support special symbols can simplify learning about your Commodore computer by providing a faster method to communicate with the disk.

Chapter 7 contains detailed instructions concerning the use of these special symbols and their limitations.

DISK MAINTENANCE COMMANDS

The following disk commands permit the user to perform file manipulation and disk maintenance.

	COMMAND	FUNCTION	BASIC 4.0 DIRECT COMMAND
	NEW	Formats a disk	HEADER
	INITIALIZE	Prepare diskette for use	----
Diskette Level	LOAD"$0"	Read disk directory	DIRECTORY
	VALIDATE	Reconstruct Block Availability Map (BAM)	COLLECT
	COPY	Copies files (optional concatenation)	COPY CONCAT
File Level	RENAME	Renames a file	RENAME
	SCRATCH	Erases a file	SCRATCH

NOTE

Diskette commands can be transmitted to the disk by PRINT# commands or through the abbreviated commands of DOS support. The examples in this chapter assume that a file has been opened with the OPEN 15,8,15 command. If the error message ?FILE OPEN ERROR appears upon typing the OPEN command, it means that the logical file was opened but had not been properly closed. This error condition will automatically close the file; to recover, retype the OPEN command.

NEW

Each time a diskette is placed in the drive, both the diskette and the drive must be prepared for use. A previously unused diskette must first be formatted in the soft-sector format recognized by your particular disk drive. This may be accomplished by use of the NEW disk command.

Before using the NEW command to format the diskette and initialize the disk drive, enter the command:

```
PRINT#15,"commandstring"
```

where 15 is the logical file number of a file which has been opened to the disk command channel (primary address 8, secondary address 15).

The format for NEW is:

```
"NEWdr:fn,xx"
```

or

```
"Ndr:fn,xx"
```

Where: dr = the drive number, 0

fn = the file name you wish to assign to the disk. It may be up to 16 characters long.

xx = a unique two-character, alphanumeric identifier supplied by the user.

The NEW command (with ID specified) is used on an unformatted diskette or one which the user wishes to reformat. NEW creates the block headers, writing the sync characters, disk ID, and track and sector numbers at the beginning of each block. The directory header and the BAM are created and the diskette is made ready to accept data. The command may be used on an already formatted diskette (with no ID specified) to clear the disk directory and reinitialize the BAM, deallocating all blocks on the diskette. The time involved in reformatting without an ID is much less than formatting with an ID.

Example 1:
```
OPEN15,8,15
PRINT#15;"N0:TEST DISK,88"
```

These commands will open the command and error channel to the disk drive and format a disk, giving it a disk identifier of 88.

Here's an example of reformatting a diskette using the NEW command and no disk ID.

Example 2:
```
OPEN1,8,15
PRINT#1,"N0:NEWNAME"
```

The diskette will be assigned the name "NEWNAME" and the directory and BAM will be cleared. This procedure will work only if the diskette has been formatted.

The NEW disk command SHOULD NOT be confused with the NEW command in BASIC. The latter will delete the program currently in memory and clear all variables before entering a new program.

HEADER (BASIC 4.0 DIRECT COMMAND)

The HEADER command has the same effect as NEW command but is reserved for those computers using BASIC 4.0. Since formatting destroys all data previously stored, the HEADER commands has a built-in safety feature that queries the user: ARE YOU SURE? A positive response to this question permits formating to take place while a negative response aborts the operation.

The format for the HEADER COMMAND IS:

HEADER"fn",Ddr,Ixx,Udn

Where: fn = file name supplied by user but limited to 16 characters.

dr = drive number, 0

xx = a unique two character alphanumeric identifier (ID) supplied by user.

(same parameters as used by NEW command)

dn = device number (Defaults to 8)

Initializing the 2031

The 2031 uses a hardware feature to detect the removal or insertion of a diskette, so it is not necessary to initialize since this is an automatic function.

THE DIRECTORY

Confirm that the newly formatted disk has the correct ID and disk name by using one of the following methods to list the directory. The directory display includes the following information:

* Disk Name
* Disk ID
* DOS version number
* File Name
* File type
* Number of blocks used
* Number of available (free) blocks

There are two methods available to all users for listing the directory. The first method illustrates the listing procedure using LOAD and the second, the listing procedure in BASIC 4.0 using the DIRECTORY command.

LOAD$

This procedure will destroy any program currently in computer memory when the directory is LOADed. (Refer to the description of the >$ DOS Support command in Chapter 7 which is a non-destructive directory display procedure.)

STEP 1: Place a formatted disk in the drive.
STEP 2: Type: LOAD"$0",8 (or LOAD"$",8) then press RETURN.

The screen displays:

```
LOAD"$0",8
SEARCHING FOR $0
LOADING
READY
```

STEP 3: Type: LIST

The drive directory will be displayed.

DIRECTORY (BASIC 4.0 DIRECT COMMAND)

This command will display the directory without disturbing the content of the memory. You may type: DIRECTORY D0 using the full word spelling but the preferred short format is illustrated which uses the first two unshifted characters followed by a shifted R.

To display the directory, type: diRd0 ON Udn and press RETURN.

If using upper case display, type: DI-D0 ON Udn and press RETURN.

The - in this example represents the upper case display for a shifted R. See your BASIC 4.0 reference manual for a complete description of this BASIC command.

Printing The Directory

Quite often, it becomes convenient to affix a diskette directory listing directly on the protective jacket. This permits the user to scan the printed directory listing without having to insert the diskette into the drive to obtain this information. Should you desire to print the directory, place the diskette in the drive and enter the following commands:

LOAD"$0",8	Loads the directory
OPEN 4,4:CMD4	Opens device 4 (printer) and changes the primary output device to 4.
LIST	Prints the directory.
PRINT#4:CLOSE4	Returns output to the screen and closes the file.

VALIDATE

The VALIDATE command traces through each block of data contained in all files on the diskette. If this trace is successful, a new BAM is generated in the disk memory and written to the diskette. Any blocks which have been allocated but are not associated with a file name, as in the case of direct access files will be freed for use. This will not affect relative files created using the BASIC 4.0 DOPEN command.

In addition to reconstructing the BAM, VALIDATE deletes files from the directory that were never properly closed. If a READ error is encountered during a VALIDATE, the operation aborts and leaves the diskette in its previous state. If a VALIDATE error does occur, you must re-initialize before proceeding.

The format for VALIDATE is:

```
PRINT#15,"VALIDATEdr"
```

Where: dr = drive number 0

NOTE

You may abbreviate VALIDATE to V.

```
Example:    OPEN1,8,15
            PRINT#1,"V0"
                 or
            PRINT#1,"VALIDATE 0"
```

COLLECT - (BASIC 4.0 Direct Command)

The COLLECT command in BASIC 4.0 performs the same function as VALIDATE. Either command will accomplish the following:

* Recreate a Block Availability Map according to valid data on disk
* Delete files from the directory which were never properly closed OPENed but never CLOSEd)

The format for COLLECT is:

```
COLLECT Dx ON Udn
```

Where: x = drive number 0

```
Example:    COLLECT D0 ON Udn
```

Verifies the drive (drive defaults to 0)

COPY

The COPY command allows you to reproduce files on the diskette or to create multiple copies (under different names). This command can also be used to concatenate data files. Up to four files can be concatenated into the destination file. The COPY command may be abbreviated with a C.

COPY disk command can be formatted three ways depending upon application.

To copy a single file: PRINT#1fn,"COPYddr:dfn=sdr:sfn"

or

PRINT#1fn,"Cddr:dfn=sdr:sfn"

NOTE

The source (sdr) and the destination (ddr) drives are the same.

To concatenate and copy PRINT#1fn,"Cddr:dfn=sdr:sfn,sdr:sfn...

Where: ddr = is the destination drive (0).

dfn = is the destination file name. This name must be different than the old file name since both files will reside on the same diskette.

In example 1, a file is reproduced on the diskette. In example 2, two files are concatenated into one file. An error message (FILE EXISTS) will be generated in the disk unit if a file to be copied already exists on the diskette.

Example 1: PRINT#1,"C0:ACCT1=0:ACCT"

A file is reproduced on the diskette.

Example 2: PRINT#1,"C0:JDATA=0:ADATA,0:BDATA"

Two files are concatenated into one file. Note that file names should be short since the maximum length of a disk command string is 58 characters.

COPY - (BASIC 4.0 Direct Command)

The COPY direct command in BASIC 4.0 performs the same function as COPY disk command and its format is also dependent upon application.

Use this format to copy a single file:

```
COPY Dsdr,"sfn" to Dddr,"dfn" ON Udn

Where sdr = the source diskette (0)

      ddr = destination diskette (0)

      fn = file name
```

Again, notice that the source (sdr) and the destination (ddr) are the same.

CONCAT (BASIC 4.0 Direct Command)

The CONCAT direct command is BASIC 4.0 permits the user to concatenate files.

The format for CONCAT is:

```
CONCAT Dsdr,"sfn" TO Dddr,"dfn" ON Udn
```

The file named dfn on drive ddr will contain the contents of both dfn and sfn after the concatenation. For example:

```
CONCAT D0,"YOURFILE" TO D0,"MYFILE"
```

This will result in MYFILE, containing the data from the old MYFILE, and YOURFILE being concatenated. YOURFILE will remain unchanged.

NOTE

The concatenation feature of COPY disk command and CONCAT direct command are valid only for DOS 2.

For further information regarding use of BASIC COPY and CONCAT commands, refer to the Commodore BASIC 4.0 Reference Manual.

RENAME

The RENAME command renames an existing file. A file can not already exist with the file name specified in the command or the FILE EXISTS error message will be generated.

The format for RENAME is:

PRINT#1fn,"RENAMEdr:nfn=ofn"

Where: dr = the disk drive on which the diskette is located (0).
nfn = the new name of the file.

ofn = the old name of the file.

1fn = a logical file number. You assign this number arbitrarily and it may be any whole number between 1 and 255.

NOTE

The letter R is a legal abbreviation for RENAME.

RENAME - (BASIC 4.0 Direct Command)

The RENAME direct command in BASIC 4.0 performs the same function as RENAME disk command.

The format is:

RENAME Ddr,"ofn" TO "nfn" (Ddr=0) ON Udn

NOTE

Close any open files before using the RENAME command since the disk will not execute this command on any active files.

For further information on the RENAME command, please refer to the Commodore BASIC 4.0 Reference Manual.

SCRATCH

The SCRATCH command erases unwanted files from the specified diskette and its directory. You can erase one file, several files, or all the files on a diskette.

The format for SCRATCH is:

PRINT#1fn,"Sdr:fn,dr:fn...dr:fn"

Where: dr = is the disk drive to be searched (0).

fn = is the name of the file to be erased.

To erase one file, enter the entire name of the file:

```
      Example:          PRINT#1,"S0:ACCT"
```

To erase several files with unrelated names, enter the entire name of each file to be deleted:

```
      Example:          PRINT#1,"S0:ACCT,0:CUSTOMER,0:INV"
```

To erase several files at one time where names have something in common, refer to the rules in APPENDIX B concerning pattern matching.

You may erase all files on a diskette using pattern matching as in the following example:

```
      Example:          PRINT#1,"S0:*"
```

SCRATCH - (BASIC 4.0 Direct Command)

SCRATCH direct command in BASIC 4.0 performs the same function as SCRATCH disk command.

```
The format is:    SCRATCH Ddr,"jn" ON Udn

      Where:      dr = drive number (0)
                  fn = filename of file to be scratched
                  dn = device number (Defaults to 8)
```

Pattern matching rules may be used with this command. As with the HEADER command, there is a built-in safety feature that queries the user: ARE YOU SURE? A positive response permits the file to be SCRATCHed while a negative response aborts the operation.

For a complete description of the SCRATCH direct command, please refer to your BASIC 4.0 reference manual.

Chapter 4

BASIC COMMANDS FOR DATA HANDLING

BASIC COMMANDS ASSOCIATED WITH FLOPPY DISK DRIVES

The BASIC commands described in this chapter, allow the user to communicate with and transfer data to and from the disk drive.

These commands are available for ALL versions of Commodore BASIC:

```
OPEN1fn,8,sa,"dr:fn,t,r/w"    VERIFY"dr:fn",8
CLOSE1fn                      PRINT#1,fn
LOAD"dr:fn",8                 GET#1fn
SAVE"dr:fn",8                 INPUT#1fn
```

These commands are available ONLY in BASIC 4.0:

```
DOPEN#1fn,"fn"                DSAVE"fn"
DCLOSE#1fn                    RECORD#1fn,R,B
DLOAD"fn
```

Where: lfn = logical file number (any number between 1 and 255)
fn = file name supplied by user
x = dr = disk drive number: both Dx and dr default to 0
8 = device number (8 for disk, 2 for second cassette, 4 for printer)
sa = secondary address
t = file type

All upper-case characters shown in format are essential for the proper execution of a command and must by typed by user. These commands are entered via the keyboard using unshifted characters only. On the CBM Business Model they will appear in lower case.

BASIC 3.0 commands are upward compatible with BASIC 4.0 commands. Each command will be defined along with a brief example to illustrate their use As soon as your dual drive floppy disk is attached to your computer and has passed the performance test, we encourage you to try the examples and procedures.

SAVE and DSAVE (Writing A Program To A Diskette)

If a program is in computer memory, it can be moved to a diskette for storage. This is accomplished with the SAVE (any Commodore BASIC) or DSAVE (BASIC 4.0) commands.

Any data transferred with the SAVE or DSAVE commands are automatically designated by the DOS as a program (PRG) file. Both commands transfer PRG files from the computer's memory to the specified diskette. You must specify the drive number, the program name, and the device number. The device number will default to device 1 which is the tape unit if it is not specified.

The format of SAVE is:

```
SAVE"dr:fn",dn
```

Where: dr = is the disk drive number. It must be 0.

fn = is any file name of 16 characters or less you wish to assign to the file to be transferred to the diskette. Blanks are counted as characters.

dn = is the device number (in this case 8).

This following example illustrates creating a one line program, SAVEing it on the diskette under the name TESTPROG, and VERIFYing that it is resident on disk.

Example:

```
10?"THIS IS A TEST"
SAVE"0:TESTPROG",8
VERIFY"0:TESTPROG",8
```

The DSAVE command performs the same function as SAVE, but is valid only with a Commodore disk system and BASIC 4.0.

The format is:

DSAVE"fn"Ddr On Udn

This command will save a file named "fn" on the floppy disk in drive 0. The file name, "fn", may be any name of 16 characters or less.

LOAD and DLOAD (Reading A Program From A Diskette)

A program stored on diskette may be loaded into memory using the LOAD (any Commodore BASIC) or DLOAD (BASIC 4.0) commands.

The LOAD and DLOAD commands transfer PRG files from the specified diskette to the computer's memory. You must specify the drive number, the program name, and the device number. The device number will default to unit 1 which is the cassette unit.

The format for LOAD is:

LOAD"dr:fn",dn

Where: dr = the drive number from which you are loading data. It must be 0.

fn = the file name prevously specified in the SAVE command and/ or stored in the disk directory.

dn = the device number (in this case 8)

The following example illustrates how a program is loaded from the diskette into the computer memory, then executed. To do this example, first type NEW and depress RETURN key to clear your computer's memory so that you can see that is really works. Don't confuse the NEW command in BASIC with the NEW disk command used to format your disk.

Example:

```
LOAD"0:TESTPROG",8
READY.
RUN
THIS IS A TEST
```

The DLOAD command transfers performs the same function as LOAD, but is specifically designed for a Commodore disk unit using BASIC 4.0. The device number will default to 8 if not specified. The drive number will default to 0 if not specified.

```
DLOAD"fn",Ddr
```

A successful LOAD or DLOAD closes all open files. Therefore you must give a new OPEN command in order to continue communicating with the disk drive command and error channel.

VERIFY

The format for VERIFY command is:

```
VERIFY"dr:fn",8
```

This command verifies that a file named "fn" stored on a floppy disk contains the same information which is stored in the computer's memory. This command is the same as the VERIFY command used with the tape cassette. Once again, dr refers to the drive number (zero). Note that the format for this command specifies that the drive number be placed before the filename. The 8 at the end of this command is the device number (8 for disk, 2 for second tape cassette).

STEP 1: Write a short program and save it on a diskette in drive 1 under the name "test" using the procedure described under the section on SAVE.

NOTE

It is important that the program in memory is not changed in any way between the save and verify operations.

STEP 2: Type: VERIFY"0:TEST",8

Once verified, the screen displays:

```
VERIFY "0:TEST",8
SEARCHING FOR 0:TEST
VERIFYING
OK

READY.
```

If a verify error occurs reSAVE the program and verify it again.

VERIFY may also be used in the format:

```
VERIFY"*",8
```

In order to perform verification of the last file saved without re-typing the filename, confirm by following these steps:

STEP 1: Write a short program and save it.

STEP 2: Clear the screen.

STEP 3: Type VERIFY"*",8

The VERIFY function will be performed comparing the last file saved to the contents of memory.

OPEN

This command sets up a correspondence between a logical file number and a file which exists on disk. It also reserves the buffer space within the disk unit for operations on the file being opened.

The format of the complete OPEN command is:

OPENlfn,dn,sa,"dr:fn,ft,mode"

Where: lfn = the logical file number

dn = the device number; in this case 8

sa = the secondary address. It may be any number from 2 to 14 and may be used either for input or output as specified in mode. See note below.

dr = the drive number: 0

fn = the name of the file.

ft = the file type. It may be SEQ (for sequential), USR (for user), REL (for relative) or PRG (for program).

mode describes how the channel is to be used. It may be either READ (R) or WRITE (W).

NOTE

Secondary address 15 is the command and error channel and has special uses which are discussed in subsequent chapters. Secondary addresses 0 and 1 are reserved by the operating systems (BASIC and DOS) for LOADing and SAVEing programs.

Examples: OPEN2,8,2,"0:FILE1,SEQ,WRITE"

OPEN3,8,9,"0:TESTDATA,PRG,WRITE"

```
OPEN8,8,8,"0:NUM,USR,READ"
```


The contents of an existing file may be replaced by preceding the drive number with an at sign (@) in the OPEN command.

```
OPEN3,8,5,"@0:JDATA,USR,WRITE"
```

If the specified file does not exist, then normal OPENing procedures are illustrated in these examples:

```
Example 1:  FL$="0:FILEA,SEQ,READ"
            OPEN1,8,14,FL$

Example 2:  FL$="0:FILEA"
            OPEN1,8,14,FL$+",SEQ,WRITE"
```

The preceding methods are convenient when it is necessary to open several channels to the same file name.

DOPEN

The DOPEN command is available only to BASIC 4.0 users. DOPEN may be used to create relative files or sequential files of fixed length.

The format of DOPEN is:

```
DOPEN#lfn,"fn",Ddr,Lrl,(,ONUdn)(,W)
```

Where: lfn, fn, and dr are the same as defined for OPEN.

Lrl defines the record length as equal to rl (relative files only)

ONUdn specifies the device number equal to dn (with default device being 8)

W may be specified to mean write mode. If W is not specified for sequential files, the file will be opened to read.

CLOSE

The CLOSE command closes a file opened by the OPEN comand. Its format is:

```
CLOSE lfn
```

Where: lfn = the logical file number of a file opened by the OPEN command.

Always close a file after working with it. You are not allowed to have more than ten open files in the computer and five in the disk drives, so it is prudent to make a habit of closing files as soon as possible. This way you will always have the maximum number of files available for use.

DCLOSE

The DCLOSE command is available only to BASIC 4.0 users. This command closes files opened with the DOPEN command.

The format for DCLOSE is:

```
DCLOSE#lfn
```

Where: lfn = the logical file number of the file to be closed.

The DCLOSE command may also be used in this format:

```
DCLOSE ONUdn
```

Where: dn = the device number of the disk unit (defaults to 8).

When used in this form, DCLOSE command closes all active disk files on the specified unit.

The following examples illustrate some applications of DCLOSE command:

Example 1: DCLOSE

Close all files currently OPEN.

Example 2: DCLOSE#5

Close only logical file 5.

CLOSING THE COMMAND CHANNEL

Closing the command channel closes all channels associated with the disk drive. No other part of the logical file environment is affected. That is, the computer does not recognize that the files have been closed.

The following example illustrates a situation in which several channels are closed down by a single CLOSE command.

Example:

```
      OPEN1,8,15                          The command channel is opened.

      OPEN3,8,2,"0:FILE1,SEQ,WRITE"       Data Channels are opened for
      OPEN4,8,2,"0:FILE2,SEQ,WRITE"       writing.

      PRINT#3,"IMPORTANT DATA"
      PRINT#4,"MORE DATA"

      OPEN3,4                             A channel is opened to the printer
                                          by mistake.

      ?FILE OPEN ERROR?                   An error message is displayed on the
                                          screen.
      READY.
```

Since there was an error, all logical files in the computer are closed, but the channels in the disk drive are still open. To close the disk channels, type:

```
      OPEN1,8,15
      CLOSE1
```

Now all data channels in the disk drive are properly closed.

CLOSING THE DATA CHANNEL

The CLOSE command closes a file an the data or command channel associated with it. When ever you close a file opened with a write channel, the closing of that file writes the final block of data to the disk and updates the disk directory. When you close a file opened with a read channel, that channel is simply closed down.

NOTE

When a drive is initialized with INITIALIZE, NEW or VALIDATE, all channels associated with that drive are deleted. These commands should not be executed when there are any files open since the files will be disrupted.

PRINT#

The PRINT# command transmits a disk command string to the drive.

The format of PRINT# is:

```
PRINT#lfn,"commandstring"
```

Where: lfn = a file previosly opened using secondary address 15

"commandstring" = disk handling or disk file handling commands. These disk commands are discussed in detail in Chapter 3 of this manual.

PRINT# may also be used to transmit data to previously-opened sequential or relative file. A semicolon must be used as a terminator for each PRINT# statement when using BASIC 3.0 to avoid sending extraneous line feeds to the diskette. These characters are written to the diskette by the BASIC PRINT# routine as part of the data terminator. It is important to be aware of this fact because the carriage return alone is seen as a terminator by the DOS. The line feed is then stored in the file as the first character in the next record. To avoid this, use the following format:

Example: PRINT#2,"JONESABC";CHR$(13);

The CHR$(13) is the carriage return necessary for the proper termination of the record on the disk. When this record is input, the result will be JONESABC which is the desired result.

BASIC 4.0 users do not need to follow this procedure (though no harmful effects will result from it). In BASIC 4.0, any file opened with lfn less than 128 will automatically suppress the line feed.

The following format may then be used:

```
PRINT#lfn,A$
```

This will produce the desired value of A$ for the record, and will not interfere with the next record.

Several variables may be written to the disk at the same time.

The format:

```
PRINT#lfn,A$,B$,C$
```

will result in a single variable (A$+B$+C$) being retrieved by the input command.

The format:

```
PRINT#lfn,A$CHR$(13)B$CHR$(13)C$
```

will result in the variables A$, B$, and C$ being separated by carriage returns, and they may then be input as separate variables.

INPUT#

The INPUT# command is used to transfer information from an IEEE device such as the disk drive into computer memory. INPUT# is valid only when used in a program and only when referencing a logical file that has been OPENed for input.

The format for INPUT# is:

INPUT#lfn,A$ or INPUT#lfn,A

Where: lfn = a file previously opened using secondary address 15

A$ = a string variable which will contain the data transferred.

A = a numeric variable which will contain the date transferred.

INPUT# may also be used to transfer several strings of data at one time:

INPUT#lfn, A$,B$,C$

Where: A$, B$, C$ will contain the data transferred from the disk.

In this format, the data strings must have been separated by carriage returns (CHR$(13)) at the time they were written tho the disk in order to be retrieved separately. No single string may contain more than 80 characters if it is to be INPUT.

EXAMPLE 1:

```
*
*
20 INPUT#2,A
*
```

Input the next data item which must be in numeric form and assign the value to variable A.

EXAMPLE 2:

```
*
*
10 INPUT#8,A$
*
```

Input the next data item as a string and assign it to variable A$.

EXAMPLE 3:

```
*
*
60 INPUT#7,B,C$
*
```

Input the next two data items and assign the first to numeric variable B and the second to string variable C$.

For strings longer than 80 characters, the GET# command must be used.

GET#

The GET# command is used to transfer individual bytes of information from an IEEE device such as the disk drive into computer memory. GET# is valid only when used in a program and only when referencing a file that has been OPENed.

The format for GET# is:

```
GET#lfn,A$
```

Where: lfn = a file previously opened using secondary address 15

A$ = a string variable which will contain the data transferred.

GET# may also be used to transfer several bytes of information, which is useful for retrieving strings which have been written to the disk in a format which is unacceptable of the INPUT# command (strings longer than 80 characters).

For example:

```
10 AA$=""
20 FOR I=1 TO 254
30 GET#lfn,A$
40 AA$=AA$+A$
50 NEXT
```

is a program segment which would result in a string of length 254 being transferred from the disk (logical file number lfn) to the computer memory and stored in the variable AA$.

RECORD#

The RECORD# command is used prior to a PRINT#, INPUT#, or GET# in order to position the file pointer to the desired record (and byte) of a relative file. For example, if record pointer is set beyond the last record and PRINT# is used, the appropriate number of records are generated to expand the file to the desired record.

RECORD# is available only to users equipped with BASIC 4.0. The format is:

```
RECORD#lfn,r,b
```

Where: lfn = a logical file number of a file previously opened with the DOPEN command

r = the desired record number. r may be either a variable name or value, however if r is a variable name, it must appear enclosed in parenthesis. I<=r<=65535

b = the byte position desired within the record. Bye positioning is optional. 1<=<=254

The following example illustrates how RECORD command is used with INPUT#:

Example:

```
10 RECORD#1,120
```

Using the RECORD command to select the record.

```
20 INPUT#1,A$
```

Input the next data item as a string and assign it to variable A$.

A detailed example of the usage of the RECORD command for relative file manipulation is found in Chapter 6.

QUICKLOAD FEATURE (BASIC 4.0)

This command feature is valid with DOS 2 and BASIC 4.0. It loads the first file on the disk into memory. To ensure that the first program on the diskette is accessed, the command must be the first disk command given after a cold start.

STEP 1: Turn the computer OFF, then ON.

STEP 2: Make sure the disk contains a program as the first file, and the drive door is closed.

STEP 3: Simultaneously press the SHIFT and RUN/STOP keys.

The computer will initialize the disk, search for the first program, and load it.

```
*** commodore basic 4.0 ***

31743 bytes free


ready
dL"*

searching for 0:*
loading
ready.
run
```

When using this feature the computer will automatically execute the DLOAD and RUN commands and it is not necessary to enter either command.

MOVING A TAPE PROGRAM TO DISK

This example illustrates a session with the computer, a tape cassette and a disk drive. The purpose is to copy a cassette program to a diskette. The program is then read from the diskette to the computer's memory and printed. It is assumed that the BASIC program was previously stored on the cassette.

Example:

```
LOAD"DEMO"

PRESS PLAY ON TAPE #1          Load the file from the cassette tape to the
                               computer's memory.

OK

SEARCHING FOR DEMO
FOUND DEMO
LOADING
READY
SAVE"0:DEMO",8                 Create a program file containing the program
VERIFY"0:DEMO",8               on diskette.
READY.

NEW                            Erase everything from memory. (The NEW
                               command in BASIC will clear memory; the
                               NEW disk command will format a disk.)

LOAD"0:DEMO",8                 Load the program back into the computer's
SEARCHING FOR 0:DEMO           memory.
LOADING
READY.

RUN                            Run the program to verify it has been loaded.
```

Chapter 5

ADVANCED DISK PROGRAMMING

This chapter provides detailed information about DOS structure and disk utility commands. The utility commands provide the programmer with low-level functions that may be used for special applications such as special disk handling routines and random access techniques.

COMMODORE DISK OPERATING SYSTEM (DOS)

The DOS file interface controller is responsible for managing all information between the disk controller and the IEEE-488 bus. Most disk I/O is performed on a pipelined basis, resulting in a faster response to a requested operation.

The file system is organized by channels which are opened with the BASIC OPEN statement. When executed with the OPEN statement, the DOS assigns a workspace to each channel and allocates either one or two disk I/O buffer areas. If either the workspace or the buffer is not available, a NO CHANNEL error is generated. The DOS also uses the channel structure to search the directory, and to delete and copy files.

The job queue is the vital link between the two controllers. Jobs are initiated on the file side by providing the disk controller with sector header and type of operation information. The disk controller seeks the optimum job and attempts execution. An error condition is then returned in place of the job command. If the job is unsuccessful, the file side re-enters the job a given number of times, depending upon the operation, before generating an error message.

The secondary address given in the OPEN statement is used by DOS as the channel number. The number the user assigns to a channel is only a reference number that is used to access the work areas, and is not related to the DOS ordering of channels. The LOAD and SAVE statements transmit secondary addresses of 0 and 1, respectively. The DOS automatically interprets these secondary addresses as LOAD and SAVE functions. Unless these functions are desired when opening files, avoid secondary addresses of 0 and 1. The remaining numbers, 2 through 14, may be used as secondary addresses to open up to five channels for data. Secondary address 15 is used to access the command channel for error messages, INPUT and GET statements and for "memory" commands.

DISK UTILITY COMMAND SET

The disk utility command set consists of the following commands:

Commands	**Abbreviations**	**General Format**
BLOCK-READ	B-R	"B-R:"ch;dr;t;s
BLOCK-WRITE	B-W	"B-W:"ch;dr;t;s
BLOCK-EXECUTIVE	B-E	"B-E:"ch;dr;t;s
BUFFER-POINTER	B-P	"B-P:"ch;p
BLOCK-ALLOCATE	B-A	"B-A:"dr;t;s
BLOCK-FREE	B-F	"B-F:"dr;t;s
memory-write	M-W	"M-W"adl/adh/nc/data
memory-read	M-R	"M-R"adl/adh
memory-execute	M-E	"M-E"adl/adh
USER	U	"Ui:parms"

Where: ch = the *channel number* in DOS: identical to the secondary address in the associated OPEN statement.

dr = the *drive number*: 0

t = the *track number*: 1 through 35

p = the *pointer position* for the buffer pointer.

adl = the *low byte* of the address*.

adh = the *high byte* of the address*.

nc = the *number of characters*: 1 through 34.

data = the actual data in hexadecimal. This is transmitted by using the CHR$ function, i.e. CHR$(1) would send the binary equivalent of hexadecimal 01, (decimal 1).

i = the *index* to the User Table.

parms = the *parameters* associated with the U command. (optional).

The values used in conjunction with the memory commands exist as hexadecimal values and must be transmitted as CHR$(n), where n is the decimal equivalent of the desired hexadecimal value.

NOTE

When using variables, the format must have only the command in quotes. For example:

```
"B-R:"ch;dr;t;s      correct
"B-R:ch,dr,t,s"      incorrect
```

To avoid confusion, it is good practice to use this format when using variables or constants.

As implied in the preceding format, these commands may be abbreviated to the first character of each of the key words. Abbreviations only are accepted for those commands shown in lower case. The parameters associated with each position if a colon is not present. The example following shows four ways that the same block-read command may be given.

Example:

```
"BLOCK-READ:"2;0;4;0

"B-R"2;0;4;0

"B-READ:"2;0;4;0
```

Parameters following the key words within quotation marks may be separated by any combination of the following characters:

Character Name	**Keyboard Representation**
Skip	<cursor right>
Space	Space bar
Comma	,

The use of these characters permits sending both ASCII strings and integers.

Parameters not within the confines of quotation marks should be separated by semicolons (;).

In the following discussions, a PRINT# is assumed in all examples.

BLOCK-READ

This diskette utility command provides direct access to any block on a diskette in either disk drive. Used in conjunction with other block commands, a random access file system may be created through BASIC. This command finds the character pointer in the 0 position of the block. When a character in this position is accessed with GET# or INPUT#, an End-or-Identify (EOI) is sent. This terminates an INPUT# and sets the Status Word (ST) to 64 in the computer.

The format "B-R"ch;dr;t;s is illustrated in the following example.

Example:

"B-R"5;0;18;0

Reads the block from track 18, sector 0 into channel 5 buffer area.

After using BLOCK READ to transfer the data to the buffer, the data may be transferred to memory by INPUT# or GET# from the logical file opened to that disk channel (i.e. , using that secondary address).

The U1 command described under USER is similar to the BLOCK-READ command.

BLOCK-WRITE

When this command is initiated, the current buffer pointer is used as the last character pointer and is placed in the 0 position of the new buffer. The buffer is then written to the indicated block on the diskette and the buffer pointer is left in position 1.

The format "B-W"ch;dr;t;s is illustrated in the following example.

Example:

"B-W"7;0;35;10

Writes channel 7 buffer to the block on track 35, sector 10: BLOCK-WRITE is not available with DOS 2.

BLOCK-EXECUTE

This command allows part of the DOS or user designed routines to reside on disk and be loaded into disk drive memory and executed. B-E is really a B-R with an addition. The file Interface Controller begins execution of the contents after the block is read into a buffer. Execution must be terminated with a return from the subroutine (RTS) instruction. Future system extensions or user-created functions may implement this command.

The format “B-E”ch;dr;t;s is illustrated in the following example.

Example:

“B-E”6;0;1;10

Reads a block from track 1, sector 10 into channel 6 buffer and executes its contents beginning at position 0 in the buffer.

BUFFER-POINTER

This command changes the pointer associated with the given channel to a new value. This is useful when accessing particular fields of a record in a block or, if the block is divided into records, individual records may be set for transmitting or receiving data.

The format “B-P”ch;p is illustrated in the following example.

Example:

“B-P”2;0

Sets channel 2 pointer to the beginning of the data area in the direct access buffer.

BLOCK-ALLOCATE

The appropriate BAM is updated in the DOS memory to reflect the indicated block as allocated (used). In future operations, the DOS skips over the allocated block when saving programs or writing sequential files. The updated BAM is written to diskette upon the closure of a write file or the closure of a direct access channel.

If the block requested has been previously allocated, the error channel indicates the next available block (increasing track and sector numbers) with a NO

BLOCK error. If there are no blocks available that are greater in number than the one requested, zeroes are displayed as track and sector parameters.

The format "B-A"dr;t;s is illustrated in the following example.

Example:

"B-A"0;10;0

Requests that block (sector)0 of track 10 be flagged as allocated on the diskette.

NOTE

The error channel should always be checked when using block allocate, so that if the block is already allocated, it will not be overwritten. If the block is allocated, the error message will also indicate the next available block.

Example:

INPUT#15,EN,EM$,ET,ES

Reads the next track and sector, respectively, into ET and ES, assuming that lfn=15 has been previously OPENed to the disk error channel.

MEMORY

All three MEMORY commands are byte oriented so that the user can utilize machine language programs. BASIC statements may be used to access information through the MEMORY commands by using the CHR$ function. The system accepts only M-R, M-W, and M-E. Neither verbose spelling nor the use of the colon (:) is permitted.

Memory-Write

This command provides direct access to the DOS memory. Special routines may be uploaded to the disk drive through this command and then executed using the MEMORY-EXECUTE command or one of the USER (U) commands. Up to 34 bytes may be deposited with each use of the command. The low byte of the address must precede the high byte of the address.

The format “M-W”adl/adh/nc/data is illustrated in the following example.

Example:

“M-W”CHR$(00)CHR$(13)CHR$(4)CHR$(32)CHR$(0)CHR$(17)CHR$(96)

Writes four bytes to buffer 2 ($1200 or decimal 4608).

MEMORY-Read

The byte pointed to by the address in the command string may be accessed with this command. Variables from the DOS or the contents of the buffers may also be read with this command. The M-R command changes the contents of the error channel since it is used for transmitting the information to the computer. The next GET# from the error channel (secondary address 15) transmits the byte. An INPUT# should not be executed from the error channel after a MEMORY-READ command until a DOS command other than one of the MEMORY commands is executed.

The format “M-R”adl/adh is illustrated in the following example.

Example:

“M-R”CHR$(128)CHR$(0)

Accesses the byte located at ($0080 or decimal 128).

Memory-Execute

Subroutines in the DOS memory may be executed with this command. To return to the DOS, terminate the subroutine with RTS($60).

The format “M-E”adl/adh is illustrated in the following example.

Example:

“M-E”CHR$(128)CHR$(49)

Requests the execution of code beginning at $3180.

USER

This command provides a link to 6502 machine code according to a jump table pointed to by the special user pointer. Refer to Table 3. The second character in this command is used as an index to the table. The ASCII character 0 through 9 or A through 0 may be used. Zero sets the user pointer to a standard jump table that contains links to special routines.

The special USER commands U1 (or UA) and U2 (or UB) can be used to replace the BLOCK-READ and the BLOCK-WRITE commands.

The format for U1 is:

"U1:"ch;dr;t;s

U1 forces the character count (buffer pointer) to 255 and reads an entire block into memory. This allows complete access to all bytes in the block.

The format for U2 is:

"U2:"ch;dr;t;s

U2 writes a buffer to a block on the disk without changing the contents of position 0 as B-W does. This is useful when a block is to be read in (with B-R) and updated (B-P to the field and PRINT#), then written back to diskette with U2.

Refer to the random access example in Chapter 6 for an application of the U1 and U2 commands.

Table 3 STANDARD JUMP TABLE

User Designation	Alternate User Designation	Function
U1	UA	BLOCK-READ replacement
U2	UB	BLOCK-WRITE replacement
U3	UC	jump to $0500
U4	UD	jump to $0503
U5	UE	jump to $0506
U6	UF	jump to $0509
U7	UG	jump to $050C
U8	UH	jump to $050F
U9	UI	jump to $0065
U:	UJ	power up vector

U3 through U9 commands are user defined. The locations jumped to are located in the buffer areas of RAM and routines may be written to reside there and uploaded using the M-W command.

FILE TYPES

Program (PRG)

This type contains data encoded in CBM BASIC format. Files of this type may be read into the CBM computer with the LOAD command and written to the disk with either the SAVE command or with the "S" command (from the editor).

Sequential (SEQ) or User (USR)

This type contains data encoded in any manner. Sequential files are used by the CBM editor and are also used by various custom programs written by users.

Relative (REL)

This type contains data encoded in any manner. Associated with the relative file will also be side sectors which hold pointers to data within the relative file. (Refer to Chapter 6 for further description).

Block, Directory and BAM Formats

A data block is addressed by track and sector. A 2031 diskette contains 35 tracks (or rings) numbered 1 to 35. The number of sectors per track will vary (as illustrated in Table 4) due to differences in track circumference and recording frequency.

The 2031 maintains a system file on track 18 which contains the BAM, diskette name, ID and file directory. The BAM, resident in the first 128 bytes of sector 0, monitors available and occupied storage locations on the diskette. The last 128 bytes of sector 0 are used to store the diskette name and ID. The file directory begins on the next sector, sector 1.

Table 4 BLOCK DISTRIBUTION BY TRACK

Track number	Block or Sector Range	Total
1 to 17	0 to 20	21
18 to 24	0 to 18	19
25 to 30	0 to 17	18
31 to 35	0 to 16	17

Any block on a diskette may be examined by using the program DISPLAY T&S, provided on the TEST/DEMO diskette.

Tables 5 and 6 will assist the user in interpreting information obtained using the DISPLAY T&S program.

Table 5 2031 BAM FORMAT

Track 18, Sector 0.

Byte	Contents	Definition
0,1	18,01	Track and sector of first directory block
2	65	ASCII character A indicating DOS 2 format
3	0	Null flag for future DOS use
4-143		Bit map of available blocks for tracks 1-35*

*1 = available block
0 = block not available
(each bit represents one block)

Table 6 2031 DIRECTORY HEADER

Track 18, Sector 0.

Byte	Contents	Definition
144-161		Disk name padded with shifted spaces.
162-163		Disk ID.
164	160	Shifted space.
165,166	50,65	ASCII representation for 2A which is DOS version and format type.
166-167	160	Shifted spaces.
171-255	0	Nulls, not used.

Note: ASCII characters may appear in locations 180 through 191 on some diskettes.

Table 7 DIRECTORY FORMAT

Track 18, Sector 1

Byte	Definition
0,1	Track and sector of next directory block
2-31	* File entry 1
34-63	* File entry 2
66-95	* File entry 3
98-127	* File entry 4
130-159	* File entry 5
162-191	* File entry 6
194-223	* File entry 7
226-255	* File entry 8

* STRUCTURE OF SINGLE DIRECTORY ENTRY (page 52)

Table 8 FORMAT OF DIRECTORY ENTRY

Byte	Contents	Definition
0	128+type	file type OR'ed with $80 to indicate properly closed file. TYPES: 0 = DELeted 1 = SEQuential 2 = ProGram 3 = UseR 4 = RELative
1,2		Track and sector of 1st data block.
3-18		File name padded with shifted spaces.
19,20		Relative file only: track and sector for first side sector block.
21		Relative file only: Record size
22-25		unused.
26,27		Track and sector of replacement file when OPEN@ is in effect.
28,29		Number of blocks in file low byte, high byte.

Table 9 SEQUENTIAL FORMAT

Byte	Definition
0,1	Track and sector of next sequential data block
2-255	254 bytes of data with carriage returns as record terminators

Table 10 PROGRAM FILE FORMAT

Byte	Definition
0,1	Track and sector of next block in program file.
2-255	254 bytes of program information stored in CBM BASIC format (with keywords tokenized). End of file is marked by three zero bytes.

Figure 4. Expanded View of a Single Sector

Figure 4 illustrates an expanded view of a single sector on a diskette formatted for the 2031. In addition to other information, each sector contains a data block consisting of 256 stored characters. Blocks within the same file are linked together by means of a two-character block pointer. By pointing to the location of the next data block, block pointers enable the system to retrieve data from non-contiguous blocks. Retrieving the first data block within a file triggers a search for the next data block which, in turn, utilizes block pointers to locate related blocks until the entire file is assembled and made available for display. All PRG, SEQ, and USR files utilize this format.

Chapter 6

ADVANCED FILE MANAGEMENT

In the preceding chapters, you learned how to manipulate files on the disk, and were shown the format of commands used to create and update files. In this chapter, you will utilize these skills in a file handling application using random or relative access.

SPECIAL OPEN AND CLOSE STATEMENTS FOR DIRECT ACCESS

The BASIC statements (after initializing the disk):

```
OPEN2,8,4,"#"
```

and

```
OPEN2,8,4,"#0"
```

are examples of how to open a channel to one buffer (to be used with the block commands). In the first example, the first available buffer is allocated to channel 4. The second example is an attempt to allocate buffer 0 to the channel.

If the buffers are not available, a NO CHANNEL error condition is generated. The explicit buffer allocation can be used to reserve a buffer for position dependent code as in the case of an EXECUTE command.

Execute a GET# statement to find the number of the allocated buffer. The byte transmitted is the buffer number. A buffer number may only be obtained PRIOR TO any write or read operations to that buffer.

The CLOSE statement clears the OPENed channel and writes the BAM to the diskette that was last used by that channel. To avoid confusion, limit yourself to accessing one drive with any direct access channel.

RANDOM ACCESS EXAMPLE

Since the BLOCK-ALLOCATE command returns the next available diskette block through the error channel, it can be used in the allocation of records. This feature allows creating a random file without being concerned with the actual physical structure of the diskette. However, the allocated blocks must first be recorded in a sequential or user file in order to be referenced by the BASIC program.

The following random file example demonstrates the use of block access entitled "RANDOMEXAMPLE". The example program is built upon a relative record scheme and provides single record access through BASIC programming. Most of the programming below line 2000 is relative record access. The field accessing routines left-justify binary and alpha fields, and right-justify numeric fields.

In an actual situation, the program should generate error messages to the operator, or automatically take corrective action such as rounding numbers to fit a field. It would also be possible to add data sorts and searches as well as key fields to the program. Record size, including field markers, must be less than 254 characters. Field size is restricted to 80 characters because of the restrictions of the BASIC INPUT# statement. Longer fields could be used if the BASIC program were modified to use GET# for retrieval but that procedure would be much slower.

Two sequential files are used to support the random access file in this example. Each file bears the name of the file name given in the CREATE file code (lines 1100 to 1180) plus a six-character extension. Since primary file names are ten or less characters, the file names are padded with spaces. The two files are named FILENAME .DESCR and FILENAME .KEY01.

The descriptor (.DESC) contains information about record structure and location. The primary key file (.KEY01) contains the first field of each record and the relative record number. This example allows the random records to reside on a separate diskette from the sequential support files, thereby providing added room for random data. The OPEN code (lines 1200 to 1275) requires the disk ID of the random file disk for comparison.

NOTE

The DOS Support Program must not be in the computer when running this program.

To Create A File:

STEP 1. Insert the Demo Disk

STEP 2. Type: LOAD "RANDOM*",8 and press RETURN

This loads the Random Program

STEP 3. Type: OPEN 15,8,15 and press RETURN

This opens the command channel.

STEP 4. Remove the Demo Disk and insert a blank disk.

STEP 5. Type: PRINT#15,"N0:MAILING LIST,CS" and press RETURN

STEP 6. Type: RUN and press RETURN

The screen displays: "DO YOU WISH TO CREATE A FILE"?

STEP 7. Type: Y and press RETURN

The screen displays: "RANDOM FILE NAME"?

STEP 8. Type the file name: PHONE LIST and press RETURN

The screen displays: "KEY FILE DRIVE NUMBER"?

STEP 9. Type: 0 and press RETURN

The screen displays: "RANDOM FILE NUMBER"?

STEP 10. Type: 0 and press RETURN

The screen displays: “ENTER ID OF RANDOM DISK”?

STEP 11. Type: CS and press RETURN

STEP 12. Type: 10 and press RETURN

For this example, ten was entered since this is the MAXIMUM number of records the file can contain. If less records are needed, specify a number less than ten. The screen displays: “NUMBER OF FIELDS PER RECORD”?

STEP 13. Type: 4 and press RETURN

This is the *number* of 'items' each field contains.

The screen displays: “INPUT FIELD NAME, FIELD SIZE, FIELD TYPE”.

TYPES: 0 = BINARY, 1 = NUMERIC, 2 = ALPHA

FIELD 1?” enter: NAME,20,2 and press RETURN

FIELD 2?” enter: PHONE,15,2 and press RETURN

FIELD 3?” enter: ADDRESS,40,2 and press RETURN

FIELD 4?” enter: COMMENTS,40,2 and press RETURN

To Add A Record:

STEP 1. The screen displays: “WHOSE RECORD DO YOU WISH TO SEE”?

Press RETURN.

The screen displays: “**** ADD RECORD ****”

NAME?

STEP 2. Type: COMMODORE and press RETURN

The screen displays: “PHONE”?

STEP 3. Type: 727-1130 and press RETURN

The screen displays: “ADDRESS”?

STEP 4. Type 3330 SCOTT BLVD SANTA CLARA CA. 95051 and press RETURN

The screen displays: “COMMENTS”?

STEP 5. Type: MANUFACTURES MICROCOMPUTERS and press RETURN

The screen displays: "WHOSE RECORD DO YOU WISH TO SEE"?

Press RETURN

The screen displays: "**** ADD RECORD ****"

NAME"? Enter the desired name. For example: SMITH and press RETURN

PHONE"? Enter the phone number: 999-356-1012 and press RETURN

ADDRESS"? Enter the address: 247 MASSOL DR LOS GATOS CA. 95030 and press RETURN

COMMENTS"? Enter a comment. For example: MANUFAC-TURES PERIPHERALS and press RETURN

STEP 6. The screen displays: "WHOSE RECORD DO YOU WISH TO SEE"?

Press RETURN

The screen displays: "**** ADD RECORD ****"

NAME"? Enter the desired name. For example: JONES and press RETURN

PHONE"? Enter the phone number: 999-26-1795 and press RETURN

ADDRESS"? Enter the address: 4086 AMBER WAY SAN JOSE CA. 95117 and press RETURN

COMMENTS"? Enter a comment. For example: MANUFAC-TURES COMPUTERS and press RETURN

To See A Record:

The computer displays: "WHOSE RECORD TO YOU WISH TO SEE"?

Enter: COMMODORE and press RETURN

To Change A Record:

After displaying the record, the screen displays: “ANY MODS”?

STEP 1. Type YES and press RETURN

The screen displays: “WHICH FIELD”?

Enter the number of the field you wish to change,

STEP 2. Type: 4 and press RETURN

The computer displays that field: US HEADQUARTERS

STEP 3. Press RETURN

The screen display asks if there are: “ANY MODS”?

STEP 4. IF the record is correct, type NO and press RETURN

Getting The Directory Of Listings:

The screen displays: “WHOSE RECORD DO YOU WISH TO SEE”?

Type: /DIR and press RETURN

The computer displays the directory.

Ending The Program:

The computer displays: “WHOSE RECORD DO YOU WISH TO SEE”?

Type: // and press RETURN. The program ends.

RELATIVE FILES

Direct access of relative files is a method that allows the programmer to position to any record on the disk relative to the beginning of that file. Compare this method to the standard procedure of having to search each track and sector for the desired information. It becomes apparent that such a relative handling method would result in a great reduction in the amount of time required to find a specific record on the disk. This reduction in the amount of time required to locate and fetch a file through the application of relative file handling techniques frees the user from the major objection to using sequential disk files: excessive “look up” time.

Relative File Components

The two main components of a relative file are the side sector chain of blocks and the data block chain. Both are linked together through forward pointers similar to those used in a sequential file. Record sizes, while fixed in length, may range from one to 254 bytes. The number of records is limited to the capacity of the disk or 65,535 records.

Side Sectors – The side sectors do not contain record information, but do contain locations of the data blocks. The record size dictates where the pointer is placed when a record numbers is referenced because the record size is used in an algorithm to compute where the pointer is placed when a record number is given through the RECORD command. The side sector also contains a table of pointers to all of the other side sectors within the file. In order to move from one side sector to another, the pointer is referenced through the appropriate DOS command, and the corresponding side track and sector read into memory. By using the information contained in the referenced side sector, the data block pointer can be located and used to read in the actual data block containing the record. The relative file data block pointers in the side sectors allow the DOS to move from one record to another within two disk read commands-a considerable savings in the amount of time required to find a desired data block when compared to sequential methods.

A file may contain up to six side sectors and each side sector may contain pointers to 120 data blocks. Therefore, the largest file on the 2031 Disk would be 182,880 bytes (120 pointers/side sector*6 side sectors*254 bytes/block) which happens to be larger than the total storage capacity of that particular disk.

Relative File Expansion

To expand a relative file, a programmer may reference the last record number generated through the RECORD command and print to that particular record. The intermediate records from the point of the current end of the file to the reference record number will be automatically generated by the DOS. This includes any side sectors and all data b locks necessary to contain a file, regardless of size, but within the capacity limits of the diskette. For example, if the current size of the relative record is one data block long and the record number referenced would expand it to 125 block, then an additional side sector would be generated by the DOS since one side sector can only represent 120 data blocks. See the paragraph entitled “EXPANDING A RELATIVE FILE” for more information and an example program.

Spanning is a key feature of relative files which aids in reducing the number of disk read/write operations required to find and retrieve data. Before explaining how this feature improves time utilization efficiency, we need to examine how I/O channels are utilized by relative files.

Table 11 RELATIVE FILE FORMAT

DATA BLOCK

Byte	Definition
0,1	Track and sector of next data block.
2-255	254 bytes of data. Empty records contain FF (all binary ones) in the first byte followed by 00 (binary all zeros) to the end of the record. Partially filled records are padded with nulls (00).

SIDE SECTOR BLOCK

Byte	Definition
0,1	Track and sector of next side sector block.
2	Side sector number. (0-5)
3	Record length.
4,5	Track and sector of first side sector (number 0)
6,7	Track and sector of second side sector (number 1)
8,9	Track and sector of third side sector (number 2)
10,11	Track and sector of fourth side sector (number 3)
12,13	Track and sector of fifth side sector (number 4)
14,15	Track and sector of sixth side sector (number 5)
16-255	Track and sector pointers to 120 data blocks.

I/O CHANNELS

When a channel is opened to a previously existing file, the DOS will position to the first record provided that the given parameters match properly. The record length variable is not necessary on the OPEN if the file is already in existence, but the DOS causes a check to be made against the record size that was originally made in the DOPEN statement creating the file. If these do not match, then error 50 -record not present- will be generated.

The relative channel requires three memory buffers form the system, whereas sequential files only require two. Since there are twelve channels in the system and two of these are used in directory searches and internal functions, only three relative channels can be open at one time. The highest number of buffers that can be used is ten, which limits the total number of channels which can be open at any one time.

Spanning

If a record was found to be on the boundary between two data blocks, that is, starting in one data block and finishing in another, then the DOS would read the first segment as well as any following records in the second data block. In practice, the records of most relative data files will span across data blocks. The only exceptions are record size 1, 2, 127, and 254. These divide evenly into the 254 size of the data block and spanning is unnecessary. This method of spanning has the advantage of requiring no system memory overhead aside from that required for the side sector blocks in the relative files. When a record is written upon through the PRINT# statement, the data block is not immediately written out. It is only written out when the DOS moves beyond the particular data block in which that record resides. This can occur through successive printing to sequential records, or when positioning to another record outside of that particular block.

NOTE

Because of the spanning feature, two channels should not be open to a relative file at the same time if either channel will be writing to the same file. An update may be made in the channel's particular memory buffer area, but the change may not be made on disk until the DOS moves that particular block. There is no restriction on this, however, and in certain instances where the file is only read from, it may be advantageous to have more than one channel open to a single relative file.

Print Termination

The DOS terminates the printing of a record by detecting the EOI signal which is generated with each PRINT# statement. If the PRINT# statement goes over the maximum record size, error 51 -record overflow- will be generated. Any data overflow will be truncated to fit the number of characters specified by the record

size and the DOS will position to the next record in sequence. If the print statement contains less characters than the record size, the remaining positions within that record will be filled with nulls. Consequently, when positioning to a record for input the EOI signal is generated from the DOS to the computer when the last non-zero is transmitted. Should the programmer desire to store binary information, a record terminator such as carriage return will have to be used and the record size is increased by one character to accommodate the terminator.

While the DOS is generating new data blocks for relative files, the requested record number is compared to the number of data blocks left on the diskette. If the resulting number of data blocks is greater than what is left on the diskette, then error 52 -file too large- is generated.

CREATING A RELATIVE FILE

The following examples apply only to those users equipped with BASIC 4.0. In terms of hardware, this means a 4000 or 8000 Series PET or CBM must be used as the computer.

When a relative file is opened for the first time, the file should be initialized by the programmer to allow for faster subsequent access, and to assure that the DOS reserves sufficient space on the diskette for future data. A simple program to perform such initialization is illustrated below:

```
110 DOPEN#1,"FILE1",D0,L50
120 GOSUB 190
130 RECORD#1,100
140 GOSUB 190
150 PRINT#1,CHR$(255)
160 GOSUB 190
170 DCLOSE#1
180 END
190 IF DS<20 THEN RETURN
200 PRINT DS$
210 IF DS=50 THEN RETURN
220 STOP
```

In the preceding example, line 110 creates a file with the name FILE1 and a record length of 50.

Lines 120, 140, and 160 cause the error handling subroutine to be executed. It is good programming practice to check the error channel after each disk-related operation.

Line 130 positions the file pointer to record number 100 which does not yet exist. The message 50 RECORD NOT PRESENT will occur at this point, but should be interpreted as a warning rather than an error condition. This message is normally expected to occur as a warning when a new record is positioned to for the first time and indicates that no INPUT or GET operation should be attempted.

Line 150 causes record number 100 (because of the pointer positioning by line 130) to be written. During this write operation, the DOS detects that record 1 thru 99 do not already exist, and automatically initializes them by placing CHR$(255) in the first character.

Line 170 closes the file and causes the space to be allocated in the BAM and file directory.

Lines 190-220 are the error subroutine. If DS is less than 20, no error condition exists, so line 190 would return control to the main program. Line 200 prints the error message, and line 210 returns to the main program if the message was 50 RECORD NOT PRESENT. If some other unexpected error (such as a read error) occurs, line 220 will halt the program so the user can correct the problem.

EXPANDING A RELATIVE FILE

After the file has been initialized, data may be written to the file. Initialization of a file in this manner need be performed only once when the file is originally created. If the user wishes to expand an existing file, the same procedure would be used, with the record number (line 130 in the example) changed to be the new last record.

The following example, when with the disk containing the file FILE1 (with 100 records) in drive 0, will result in the first 100 records remaining unchanged and records containing only CHR$(255) would be generated for records 101-200.

```
110 DOPEN#1,”FILE1”,D0,L50
120 GOSUB 190
130 RECORD#1,200
140 GOSUB 190
170 DCLOSE#1
180 END
190 IF DS<20 THEN RETURN
200 PRINT DS$
210 IF DS=50 THEN RETURN
220 STOP
```

NOTE

When DOPEN is used on an existing file, specification of the record length is optional. If specified, it must match the record length set at the time the file was created or an error condition will result.

When a file is expanded in this manner, the required side sectors are also created. Side sectors are transparent to the user since they are automatically generated and accessed by the DOS.

ACCESSING A RELATIVE FILE

In order to make the relative file system practical, the user must be able to access the file for reading and writing of data. Both of these operations are simplified by relative files and both may use the RECORD command for positioning to the desired record before the operation.

To write data to a predetermined record in a file, a constant may be used in the positioning (line 130) as follows:

```
110 DOPEN#1,”FILE1”,D0
120 GOSUB 190
130 RECORD#1,25
140 GOSUB 190
150 PRINT#1,”RECORD 25”
160 GOSUB 190
170 DCLOSE#1
180 END
190 IF DS<20 THEN RETURN
200 PRINT DS$
210 IF DS=50 THEN RETURN
220 STOP
```

The resulting record would appear as follows:

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------
RECORD 25*
--------------------------------------------------
```

Where * represents a carriage return (CHR$(13)).

The following program illustrates the feature which permits access to individual bytes within a record:

```
110 DOPEN#1,”FILE1”,D0
120 GOSUB 900
130 RECORD#1,25,1
140 GOSUB 900
150 PRINT#1,”FIELD 1”
160 GOSUB 900
170 RECORD#1,25,10
180 GOSUB 900
190 PRINT#1,”FIELD 2”
200 GOSUB 900
210 RECORD#1,25,30
220 GOSUB 900
230 PRINT#1,”FIELD 3”
240 GOSUB 900
250 DCLOSE#1
260 END
900 IF DS<20 THEN RETURN
910 PRINT DS$
920 STOP
```

Lines 130, 170, and 210 cause the file pointer to be moved to different places within record number 25. The following illustration is a representation of the contents of record number 25 after the above example is executed:

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
--------------------------------------------------
FIELD 1* FIELD 2*            FIELD 3*
--------------------------------------------------
```

Where * represents a carriage return (CHR$(13)).

NOTE

It is important that the fields are written in order because writing to a byte at the beginning of the record flushes the rest of the record in memory. This means that while it is possible to position and write first to byte 1 and then to byte 20, it is NOT possible to first write byte 20 and then byte 1.

Since the carriage return is recognized as a terminator by the BASIC INPUT statement, the data may be retrieved by the following sequence:

```
110 DOPEN#1,”FILE1”,D0
120 GOSUB 290
130 RECORD#1,25
140 GOSUB 290
150 RECORD#1,25,1:GOSUB 290
160 INPUT#1,A$:GOSUB 290
170 RECORD#1,25,10:GOSUB 290
180 INPUT#1,B$:GOSUB 290
190 RECORD#1,25,30::GOSUB 290
200 INPUT#1,C$:GOSUB 290
210 DCLOSE#1
220 END
290 IF DS<20 THEN RETURN
300 PRINT DS$
310 STOP
```

Lines 160, 180, and 200 cause the stored values on disk to be read and stored in A$, B$, and C$, respectively.

It is extremely useful to be able to access a record which is determined during program operation. The following routine illustrates a procedure to query the operator for a record number and a data and to write the data to the disk file:

```
100 PRINT”TYPE RECORD NUMBER AND DATA”
105 INPUT R,D$
110 DOPEN#1,”FILE1”,D0
120 GOSUB 190
130 RECORD#1,(R)
140 GOSUB 190
150 PRINT#1,D$
160 GOSUB 190
170 DCLOSE#1
180 END
190 IF DS<20 THEN RETURN
200 PRINT DS$
210 STOP
```

Line 130 positions the file pointer to record number (R), specified by the user. Note that a variable used in the RECORD command must be enclosed in parenthesis.

Line 150 causes the data stored in D$ to be stored on the disk.

The RECORD command may be omitted if the file is to be accessed sequentially, which saves time during program execution. An example of this occurs when writing a large data base to the disk file. Assume that the program has

already dimensioned D$ as an array which contains 100 elements. These elements are to be written to the disk in records number 1 through 100 of file FILE1. This could be accomplished with the following program segment:

```
*
*
*
110 DOPEN#1,"FILE1",D0
120 GOSUB 190
130 FOR I=1 TO 100
150 PRINT#1,D$(I)
160 GOSUB 190
170 DCLOSE#1
180 END
190 IF DS<20 THEN RETURN
200 PRINT DS$
210 STOP
```

Since the record pointer is automatically set to record 1 when the file is opened, record 1 is the first record written. If no RECORD command is executed the DOS automatically positions to the next record after each PRINT. Therefore, the contents of D$ will be written to records 1 through 100 of the file.

For detailed description of related BASIC commands, refer to Chapter 4 of this manual and the BASIC 4.0 reference manual.

Chapter 7

SIMPLIFYING THE USE OF COMMODORE DISK-RELATED COMMANDS

It has been explained that all disk commands must be preceded with the BASIC PRINT# command and enclosed in quotation marks. This is true, but your computer can be programmed to perform these tasks as well as how to load and run programs stored on diskette.

LOADING THE DOS SUPPORT PROGRAM

The first file on the TEST/DEMO diskette contains the program DOS SUPPORT. This program, when loaded into memory, takes care of the tasks mentioned above. If your computer has Commodore BASIC 4.0 you can use the quick load procedure to load the DOS SUPPORT program. For those of you not equipped with BASIC 4.0, the following procedure will work:

Start with a cold start condition by resetting both the computer and disk drive and insert the TEST/DEMO disk in drive 0.

STEP 1. Type: LOAD”*”,8 and press RETURN.

The screen displays:

```
READY.
LOAD”*”,8
SEARCHING FOR *
LOADING
READY.
```

STEP 2. Type: RUN and press RETURN.

This will cause the DOS Support program to be executed. This program will relocate itself into the top of the user memory, where it will coexist with programs which are entered later. DOS support will not need to be reloaded until the computer is reset. The following special symbols, once implemented, will simplify the entry of disk commands.

USING THE DOS SUPPORT SYMBOLS: > and @

Once DOS SUPPORT is implemented, preceding disk commands with PRINT#lfn or enclosing them in quotation marks is no longer required: precede the disk command with either the greater than symbol (>) or the at-sign (@). The examples in this manual use the > symbol.

Examples:

```
>I0 is the same as PRINT#1,”I0”

>S0:FILE1 is the same as PRINT#15,”S0;FILE1”
```

The OPEN statement is NOT required before a statement.

The > symbol can also be used to load a diskette directory. Normally the directory is loaded with LOAD”$dr”,8 but this command destroys any program you might have in memory. When you use the > symbol, the directory is printed directly to the screen, thus preserving the data in the computer's memory.

Example:

```
>$0 means display the entire directory

>$0:Q* means to display all the files on the drive
```

NOTE

To avoid scrolling the directory, press the space bar to stop the listing. To continue the listing, press any key on the keyboard.

To stop a directory listing and return to BASIC, press RUN/STOP.

The third use of > is the request of error messages.

Example:

> is equivalent to:

```
10 OPEN2,8,15
20 INPUT#2,A$,B$,C$,D$
30 PRINTA$,B$,C$,D$
```

LOADING A PROGRAM WITH THE /

Use the slash (/) to load a program from the diskette. Both diskettes are searched if the drive number is not specified.

Example:

/ACCT loads the program ACCT into the computer's memory.

LOADING AND RUNNING A PROGRAM WITH UP ARROW

The up arrow (↑) loads a program from a diskette and executes it. Both diskettes are searched if necessary.

Example:

↑JDATA loads and runs the program JDATA.

SPECIAL DOS SUPPORT INFORMATION

The DOS SUPPORT program has certain limitations. These are:

1. The program must be re-accessed from the disk whenever resetting the computer.

2. DOS Support may only be used when communicating with the disk in direct mode. That is, they may NOT be used in a program.

3. The disk directory may be printed on the printer by giving these commands:

```
LOAD "$0",8
OPEN 4,4:CMD4:LIST
PRINT#4:CLOSE4
```

4. DOS support will only operate on device number 8.

CHANGING DEVICE NUMBER

Occasionally it is desirable to alter the device number (primary address) of a disk system. As assembled at the factory, each CBM Floppy Disk System has a primary address of 8. The following program will allow the change of the address to other values.

NOTE

The address change remains in effect until the system is reset either by the primary AC power switch to the disk system or to the CBM computer, or by the execution of the User Reset Vector "UJ" (Described in the manual page 47/48).

The program to change the device number (primary address) is:

```
10 OPEN15,CH,15
20 PRINT#15,"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(NA$+32)CHR$(NA+64)
```

Where: CH = current primary address (8 at power on/reset)
NA = new address (only 8-11 is allowed)

NOTE

If any address other than 8 is set into the system, all BASIC 4.0 disk commands must be suffixed with Udn parameter as the default is to device number 8.

CHAPTER 8

ERROR MESSAGES - PATTERN MATCHING FILE NAMES - DISK COMMANDS

REQUESTING ERROR MESSAGES: COMMODORE DISK DRIVES

The execution of the following programs displays the error on the computer screen and resets the device error indicator:

```
      CBM Series 2001         CBM Series 8000
      CBM Series 3000         CBM Series 4000
            with                    with
          BASIC 3.0               BASIC 4.0

      10 OPEN 1,8,15          PRINT DS$
      20 INPUT#1,A,B$,C,D
      30 PRINT A,B$,C,D
```

where A = error message number B$ = error message C = track D = sector

SUMMARY OF CBM FLOPPY ERRROR MESSAGES

```
0     OK, no error exists.
1     Files scratched response. Not an error condition.
2-19  Unused error messages: should be ignored.
20    Block header not found on disk.
21    Sync character not found.
22    Data block not present.
23    Checksum error in data.
24    Byte decoding error.
25    Write-verify error.
26    Attempt to write with write protect on.
27    Checksum error in header.
28    Data extends into next block.
29    Disk id mismatch.
30    General syntax error.
31    Invalid command.
32    Long line.
33    Invalid filename.
34    No file given.
39    Command file not found.
50    Record not present.
51    Overflow in record.
52    File too large.
60    File open for write.
61    File not open.
62    File not found.
63    File exists.
64    File type mismatch.
65    No block.
66    Illegal track or sector.
67    Illegal system track or sector.
70    No channels available.
71    Directory error.
72    Disk full or directory full.
73    Power up message, or write attempt with DOS mismatch.
74    Drive not ready.
```

DESCRIPTION OF DOS ERROR MESSAGES

NOTE

Error message numbers less than 20 should be ignored with the exception of 01 which gives information about the number of files scratched with the SCRATCH command.

20: READ ERROR (block header not found)
The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.

21: READ ERROR (no sync character)
The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/write head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.

22: READ ERROR (data block not present)
The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with The BLOCK commands and indicates an illegal track and/or sector request.

23: READ ERROR (checksum error in data block)
This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.

24: READ ERROR (byte decoding error)
The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.

25: WRITE ERROR (write-verify error)
This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.

26: WRITE PROTECT ON
This message is generated when the controller has been requested to write a data block while the write protect switch in depressed. Typically, this is caused by using a diskette with a write protect tab over the notch.

27: READ ERROR (checksum error in header)
The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.

28: WRITE ERROR (long data block)
The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad diskette format (the data extends into the next block), or by hardware failure.

29: DISK ID MISMATCH
This message is generated when the controller has been requested to access a diskette which has not been initialized The message can also occur if a diskette has a bad header.

30: SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel. Typically, this is caused by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.

31: SYNTAX ERROR (invalid command)
The DOS does not recognize the command. The command must start in the first position.

32: SYNTAX ERROR (long line)
The command sent is longer than 58 characters.

33: SYNTAX ERROR (invalid file name)
Pattern matching is invalidly used in the OPEN or SAVE command.

34: SYNTAX ERROR (no file given)
The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon (:) has been left out of the command.

39: SYNTAX ERROR (invalid command)
This error may result if the command sent to command channel (secondary address 15) is unrecognizable by the DOS.

50: RECORD NOT PRESENT
Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.

51: OVERFLOW IN RECORD
PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted to the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.

52: FILE TOO LARGE
Record position within a relative indicates that disk overflow will result.

60: WRITE FILE OPEN
This message is generated within a write file that has not been closed is being opened for reading.

61: FILE NOT OPEN
This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.

62: FILE NOT FOUND
The requested file does not exist on the indicated drive.

63: FILE EXISTS
The file name of the file being created already exists on the diskette.

64: FILE TYPE MISMATCH
The file type does not match the file type in the directory entry for the requested file.

65: NO BLOCK
This message occurs in conjunction with the B-A command. It indicates that the block to the allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use.

66: ILLEGAL TRACK AND SECTOR
The DOS has attempted to access a track or sector which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.

67: ILLEGAL SYSTEM T OR S
This special error message indicates an illegal system track or sector.

70: NO CHANNEL (available)
The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.

71: DIR(ectory) ERROR
The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem, reinitialize the diskette to restore the BAM in memory. Some active files may be terminated by the corrective action. NOTE: BAM = Block Availability Map.

72: DISK FULL
Either the blocks on the diskette are used or the directory is at its limit 144 entries. DISK FULL is sent when two blocks are available to allow the current file to be closed.

73: DOS MISMATCH (73, CBM DOS V2.6 2031)
DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. (A utility routine is available to assist in converting form one format to another.) This message may also appear after power up.

74: DRIVE NOT READY
An attempt has been made to access the 2031 Disk Drive without a diskette present.

PATTERN MATCHING

Pattern matching of file names is available on all Commodore floppies. Pattern matching uses the question mark (?) and the asterisk (*) to perform operations on several files with similar names.

The asterisk is used at the end of a string of characters to indicate that the rest of the name is insignificant. For example:

```
           FIL*        could refer to files named
           FIL
     or    FILE1
     or    FILEDATA
     or    FILLER
     or    any other file name starting with the letters FIL.
```

The question mark may be used anywhere within the string of characters to indicate that the character in that particular position should be disregarded. For example:

```
           ?????.SRC         could refer to files named
           TSTER.SRC
     or    DIAGN.SRC
     or    PROGR.SRC
     but not SRC.FILES
```

Both the characters and the position of the characters are significant.

The question mark and asterisk may be combined in many ways:

```
           *J??????
```

does not make sense because the question marks are in an area which is insignificant (because of the asterisk).

```
           P???FIL*          will access files with the names
           PET FILE
     or    PRG FILE-32
     or    POKEFILES$$
     or    any other files starting with P and having FIL in positions 5-7.
```

SCRATCH with pattern matching should be used carefully, since multiple files will be scratched. LOAD or DLOAD will load the first file which fits the pattern matching. OPEN or DOPEN with pattern matching may be used to open an existing file, in which case the first existing file encountered which fits the description will be opened. However, OPEN or DOPEN should not be used with pattern matching when creating a new file. Never use RENAME, SAVE, DSAVE, or COPY for pattern matching since an error condition will result, if attempted.

USER'S QUICK REFERENCE: DISK COMMANDS

The user's quick reference guide will assist the user in becoming familiar with the various commands used in both BASIC 3.0 and BASIC 4.0, and with the DOS SUPPORT utility as well as with all Commodore disk units.

In order to make BASIC 4.0 easier to use, disk commands have been incorporated into the language. For example, with BASIC 4.0:

* DSAVE and DLOAD commands eliminate the need to specify device number each time you store and retrieve disk files.
* Directory display is now a one-step procedure and no longer interferes with the program in memory.
* It is no longer necessary to write a program to read the error channel. The variable DS$ contains the error message.
* Formatting in now a one-step procedure through the use of HEADER command.

Commands in BASIC 3.0 are upward compatible with BASIC 4.0. That is, if you are familiar with BASIC 3.0, those commands will still work on the Series 8000 Computer furnished with BASIC 4.0. All disk commands available on the 2040 are upward compatible with both the 4040 and 8050.

BASIC 3.0	**UNIVERSAL DOS SUPPORT**	**BASIC 4.0**
SAVE “dr:fn”,8	SAVE”dr:fn”,8	DSAVE,”fn”,Ddr (drive defaults to 0)
LOAD”dr:fn”,8	/dr:fn (searches both drives)	DLOAD”fn”,Ddr (drive defaults to 0)
LOAD”*”,8 RUN	/dr:fn	DLOAD”fn”,Ddr RUN
LOAD”dr:fn”,8 RUN	*	shifted RUN/STOP
LOAD”$0”,8 LIST	>$0	DIRECTORY or DI<shifted R>
destroys memory	preserves memory	preserves memory
10 OPEN1,8,15 20 INPUT#1,A,B$,C,D 30 PRINT A,B$,C,D	> return	?DS$ or ?DS (DS is number of error only)

NOTE

Assume that OPEN1,8,15 has already been typed for all of the PRINT# commands in the following formats. Commands may be spelled out or abbreviated by the first letter as illustrated.

	INITIALIZE	
PRINT#1,"Ix"	>Ix	PRINT#1,"Ix"
	VALIDATE	
PRINT#1,"Vdr"	>Vdr	COLLECT Ddr,Udn
	SCRATCH	
PRINT#1,"Sdr:fn"	>dr:fn	SCRATCH"fn",Ddr,Udn
	COPY (single file)	
PRINT#1,"Cdr:dfn= dr:sfn"	>Cdr:dfn=dr:sfn	COPY Ddr,"sfn" TO Ddr,"dfn",Udn
	CONCATENATE FILES	
PRINT#1,"Cdr:dfn= dr:sfn1,dr:sfn2,...	>Cdr:dfn=dr:sfl, dr:sfn2,...	CONCAT Ddr,"sfn" TO Ddr,"dfn",Udn
	RENAME FILES	
PRINT#1,"Rdr:dfn= sfn"	>Rdr:dfn="sfn"	RENAME Ddr,"sfn" TO "dfn",Udn
	FORMAT A DISKETTE	
PRINT#1,"Ndr: dname,xx"	>Ndr:dname,xx	HEADER "dname", Ddr,Ixx,Udn

Chapter 9

RANDOM 1.00 PROGRAM LISTING

This chapter contains a complete listing of the random access program described in chapter 6 under the heading of Random Access Example.

[Editor's Note on following program: The original was heavily crunched. I have added spaces to make the code more reader friendly.]

```
1 REM 2031 RANDOMEXAMPLE
2 REM
3 REM SUBROUTINES TO MANAGE RANDOM ACCESS FILES
4 REM VARIABLES ARE SET FROM DATA OF DESCRIPTION FILE & KEY LIST FILES...
5 REM DEFINED BY USER PROGRAM
6 REM VARIABLES SHOULD REFLECT DESIRED FILE STRUCTURE
7 REM ALL FUNCTIONS ACT UPON THE VARIABLES DEFINED BELOW
10 REM
11 REM ****************************
12 REM
15 M$=CHR$(13):REM FIELD MARKER
16 SP$="                          "+"":REM SPACE FOR PADDING
20 C0=2:       REM DIRECT CHANNEL
21 C1=3:       REM SEQUENTIAL CHANNEL
25 CC=15:      REM COMMAND CHANNEL
30 D=0:        REM CURRENT DRIVE #
31 T=0:        REM CURRENT TRACK #
32 S=0:        REM CURRENT SECTOR #
35 DD=0:       REM DESCRIPTOR DRIVE #
36 RD=0:       REM RANDOM DRIVE #
```

```
40 ID$="":            REM RANDOM DISK ID
45 NR=0:              REM # RECORDS IN R-FILE
46 CR=0:              REM CURRENT RECORD #
47 FR=0:              REM 1ST FREE RECORD UNUSED
50 NF=0:              REM # FIELDS IN RECORD
51 CF=0:              REM CURRENT FIELD #
55 RB=0:              REM # RECORDS PER BLOCK
56 RS=0:              REM RECORD SIZE IN BYTES
60 NB=0:              REM # BLOCKS IN R-FILE
65 E=0:               REM ERROR FLAG, OK=0
66 REM EN$,EM$,ET$,ES$,ET,ES  ERROR CHANNEL VARIABLES
70 EP=.5/256:         REM INTEGER CORRECTION
75 AS=0:              REM INDEEX ARRAY ADDRESSING STRTEGY
76 REM AS=0:  USE ARRAY INDEX; AS=1: T&S ARE SET, CR= RECORD OFFSET IN BLOCK
90 REM "A" VARIABLES ARE TEMPORARY
95 DN=8:OPENCC,DN,CC:  REM DN=DEVICE NUMBER
98 GOTO 2000:         REM START OF USER PROGRAM
99 REM
100 REM ********************************
101 REM RANDOM FILE DIMENSION ROUTINE
102 REM 1ST SET NR, NF, &NB
103 REM
105 GOSUB 150
110 IF FP%=-1 THEN RETURN
111 FP%=-1
115 DIM FS%(NF):      REM FIELD SIZE
120 DIM FP%(NF):      REM FIELD POSITION
125 REM               FP%(I(=SUM [FS%(I-1)]
130 DIM FT%(NF)       :REM FIELD SIZE
135 DIM FH$(NF)       :REM FIELD POSITION
140 DIM F$(NF)        :REM FIELD ARGS-ALPHA, BINARY
145 DIM F(NF)         :REM FIELD ARGS-NUMERIC
146 RETURN
150 IF IT%=-1 THEN RETURN
151 IT%=-1
155 DIM IT%(NB)       :REM TRACK INDEX ARRAY
160 DIM IS%(NB)       :REM SECTOR ARRAY INDEX
165 DIM K1$(NR)       :REM PRIMARY KEY VALUE
170 DIM RR%(NR)       :REM RELATIVE RECORD LIST PER KEY
175 RETURN
200 REM *************************
201 REM UPDATE RECORD, CR
202 REM
205 GOSUB 900
210 PRINT#CC,"U1:"C0;D;T;S
215 PRINT#CC,"B-P:"C0;RP
220 FOR CF=1 TO NF
225 GOSUB 1000:IF E THEN 1900
230 NEXT CF
235 PRINT#C,"U2:"C0;D;T;S
240 GOSUB 1000:IF E THEN 1900
245 RETURN
300 REM ****************************
301 REM READ RECORD, CR
302 REM
305 GOSUB900
310 PRINT#CC,"U1:"C0;D;T;S
315 PRINT#CC,"B-P:"C0;RP
320 GOSUB 1000:IF E THEN 1900
325 FOR CF=1 TO NF
330 GOSUB 600
335 NEXT CF
340 RETURN
400 REM *************************
401 REM UPDATE FIELD(CF) OF RECORD CR, SINGLE FIELD UPDATE
402 REM
405 GOSUB 900
```

```
410 PRINT#CC,"U1:"C0;D;T;S
415 GOSUB 1000:IF E THEN 1900
420 PRINT#CC,"B-P:"C0;FP%(CF)+RP
425 GOSUB 500  :REM UPDATE FIELD
430 PRINT#CC,"U2:"C0;D;T;S
435 GOSUB 1000:IF E THEN 1900
440 RETURN
450 REM *************************
451 REM READ FIELD(CF) OF RECORD CR, SINGLE FIELD READ
452 REM
455 GOSUB 900
460 PRINT#CC,"U1:"C0;D;T;S
470 GOSUB 1000:IF E THEN 1900
475 GOSUB 600  :REM READ FIELD
480 RETURN
500 REM *************************
501 REM UPDATE FIELD(CF), B-P IS SET
502 REM
510 IF FT%(CF)<>1 THEN 520
515 A$=RIGHT$(SP$+STR$(F(CF)),FS%(CF)):GOTO 530
520 A$=LEFT$(F$(CF)+SP$,FS%(CF))
530 PRINT#C0,A$;M$
535 RETURN
600 REM *************************
601 REM READ FIELD(CF), B-P IS SET
602 REM
610 IF FT%(CF) THEN 645
615 A1$=""
620 FOR J=1 TO FS%(CF)
625 GET#C0,A$:IF A$="" THEN A$=CHR$(0)
630 A1$=A1$+A$
635 NEXT:F$(CF)=A1$
640 GET#C0,A$:RETURN
645 INPUT#C0,F$(CF)
650 IF FT%(CF)<>1 THEN RETURN
655 F(CF)=VAL(F$(CF)):RETURN
700 REM *************************
701 REM ALLOCATE ONE BLOCK, T&S =REQUESTED TRACK & SECTOR
702 REM  RETURNED T&S ARE ALLOCATED VALUES (T=18 IS SKIPPED)
703 REM
710 GOSUB 800:IF E THEN 1900:  REM CHECK T & S
715 PRINT#CC,"B-A:"D;T;S
720 INPUT#CC,EN,EM$,ET,ES
725 IF EN=0 THEN RETURN
730 IF EN<>65 THEN 1900
735 IF ET=18 THEN T=19:S=0:GOTO 715
736 T=ET:S=ES
740 GOTO 715
750 REM ************************
751 REM FREE ONE BLOCK, T & S = TRACK  & SECTOR
752 REM
760 GOSUB 800:IF E THEN 1900:  REM CHECK T & S
770 PRINT#CC,"B-F:"D;T;S
780 INPUT#CC,EN,EM$,ET,ES
785 IF EN=0 THEN RETURN
790 GOTO 1900
800 REM ************************
801 REM CHECK MAX SECTOR
802 REM
810 IF T>35 THEN 1900
820 E=0:IF T=0 THEN T=40:GOTO 1900: REM ORIGINAL LINE READ... E=0:IFT=0THEN=40:GOTO1900
840 A3=16:IF T>30 THEN 880
850 A3=17:IF T>24 THEN 880
860 A3=19:IF T>17 THEN 880
870 A3=20
880 IF S>A3 THEN 1900
890 RETURN
```

```
900 REM ************************
901 REM SET RECORD'S TRACK, SECTOR & RECORD POINTER FROM INDEX ARRAYS
902 REM
905 D=RD
910 E=0
915 IF AS=-1 THEN RP=CR*RS+1:GOTO 950
920 RP=INT((CR-1)/RB+EP):IF RP> NB OR RP<0 THEN EN=41:GOTO 1900
930 T=IT%(RP):S=IS%(RP)
940 RP=INT(((CR-1)/RB-RP+EP)*RS*RB)+1
950 IF RP>254 THEN EN=41:GOTO 1900
960 RETURN
1000 REM ************************
1001 REM INPUT 2040 ERROR STATUS
1002 REM
1005 INPUT#CC,EN$,ET$,ET,ES
1010 EN=VAL(EN$):E=0
1015 IF EN$="00"THEN RETURN
1017 ET$=STR$(ET):ES$=STR$(ES)
1020 IF EN$<>RIGHT$("0"+MID$(STR$(EN),2),2)THEN 1070
1030 IF EN=1 THEN EM$=ET$+" "+EM$:RETURN
1035 E=E+1
1040 EM$="RVS ON"+EN$+"RVS OFF"+EM$ :REM RVS ON = REVERSE CHARACTERS, RVS OFF - TURN OFF REVERSE CHAR
1050 IF EN<30 OR EN=65 THEN EM$=EM$+" ON "+ET$+", "+ES$
1060 RETURN
1070 EM$="RVS ON SYSTEM NOT RESPONDING PROPERLY"
1080 EM$=EM$+EN$+EM$+ET$+ES$
1085 E=E+1
1090 RETURN
1100 REM *********************
1101 REM CREATE DESCRIPTOR FILE
1102 REM INPUT F$=FILENAME
1103 REM       ID$,NR,NF,FS%(),FT%(),FH$()
1104 REM       DD=DESCRIPTOR FILE DRIVE #
1105 REM       RD=RANDOM DISK DRIVE #
1106 REM DRIVES MUST BE INITIALIZED
1109 REM
1110 RS=1:D=RD
1115 FOR A0=1 TO NF:FP%(A0)=RS:RS=FS%(A0)+RS+1:NEXT:RS=RS-1
1116 RB=INT(254/RS+EP)
1120 OPEN C0,DN,C0,"#":GOSUB 1000:IF E THEN 1900
1121 GOSUB 1280
1122 PRINT#CC,"B-P:"C0;1
1123 FOR A0=1 TO RB:FOR A1=1 TO NF
1124 PRINT#C0,LEFT$(SP$,FS%(A1));M$;
1126 NEXT A1,A0
1130 NB=INT(NR/RB+EP):IF (NR/RB-NB)*RB>=1 THEN NB=NB+1
1135 T=1:S=0:GOSUB 150
1140 FOR A0=0 TO NB-1:GOSUB 710:IF E THEN 1900
1145 IT%(A0)=T:IS%(A0)=S:GOSUB 430:NEXT
1150 GOSUB 710
1152 PRINT#CC,"B-P:"C0;1
1155 PRINT#C0,NR;M$;1;M$;NB;M$;RS;M$RB;M$;NF;M$
1160 PRINT#CC,"B-W:"C0;D;T;S
1165 A$=STR$(DD)+":"+LEFT$(F$+SP$,10)++".DESCR,U,W"
1166 OPEN C1,DN,C1,A$
1167 GOSUB 1000:IF E THEN 1900
1168 PRINT#C1,ID$;M$;T;M$;S;M$
1170 FOR A0=1 TO NF:PRINT#C1,CHR$(FS%(A0));CHR$(FT%(A0));FH$(A0);M$;:NEXT
1175 FOR A0=0 TO NB-1:PRINT#C1,CHR$(IT%(A0));CHR$(IS%(A0));:NEXT
1180 CLOSE C1:CLOSE C0:RETURN
1200 REM ******************
1201 REM OPEN RELATIVE FILE
1202 REM INPUT F$= FILENAME
1203 REM       DD= DESCRIPTOR FILE DRIVE #
1204 REM       RD= RANDOM DISK DRIVE #
1205 REM DRIVES MUST BE INITIALIZED
1209 REM
```

```
1210 A$=STR$(DD)+":"+LEFT$(F$+SP$,10)+".DESCR,U,R"
```


```
1215 OPEN C1,DN,C1,A$:GOSUB 1000:IF E THEN 1900
1220 INPUT#C1,ID$,T,S
1225 OPEN C0,DN,C0,"#":GOSUB 1000:IF E THEN 1900
1226 GOSUB 1280
1227 PRINT#CC,"B-R:";C0;RD;T;S:GOSUB 1000:IF E THEN 1900
1230 INPUT#C0,NR,FR,NB,RS,RB,NF
1235 GOSUB 100:FT%(0)=T:FS%(0)=S
1240 FOR A=0 TO NF:GOSUB 1298:FS%(A0)=ASC(A$)
1245 GOSUB 1298:FT%(A0)=ASC(A$)
1250 INPUT#C1,FH$(A0):NEXT
1255 FOR A0=0 TO NB-1:GOSUB 1298:IT%(A0)=ASC$(A$)
1260 GOSUB 1298:IS%(A0)=ASC(A$):NEXT
1265 GOSUB 1000:IF E THEN 1900
1270 CLOSE C1
1275 RETURN
1280 PRINT#CC,"U1:";C0;RD;",18,0":GOSUB 1000:IF E THEN1 900
1285 PRINT#CC,"B-P:"C0;162
1286 GET#C0,A$,A1$:A$=A$+A1$;IF ID$<>A$ THEN EN=43:EM$="DISKID MISMATCH":GOTO 1900
1290 RETURN
1298 GET#C1,A$:IF A$="" THEN A$=CHR$(0)
1299 RETURN
1400 REM ************************
1401 REM CLOSE RELATIVE FILE
1402 REM INPUT: VARIABLES FROM OPEN SHOULD BE VALID
1409 REM
1410 PRINT#CC,"B-P:"C0;1
1420 PRINT#C0,NR;M$;FR;M$;NB;M$;RS;M$;RB;M$;NF;M$;
1430 PRINT#CC,"B-W:"C0;D;FT%(0);FS%(0)
1440 CLOSE C0
1490 RETURN
1900 E=E+1:RETURN
2000 INPUT"♥Q QDO YOU WISH TO CREATE A FILE Y]]]";A$:
     IF LEFT$(A$,1)="N" THEN 2100
2001 INPUT"QRANDOM FILE NAME PHONE LIST]]]]]]]]]]]]",F$
2002 INPUT"KEY FILE DRIVE NUMBER  0]]]";DD
2003 INPUT"RANDOM FILE DRIVE NUMBER  0]]]";RD
2004 INPUT"ENTER ID OF RANDOM DISK  CS]]]]";ID$:ID$=LEFT$(ID$,2)
2006 INPUT"NUMBER OF RECORDS";NR
2007 INPUT"NUMBER OF FIELDS PER RECORD";NF
2010 GOSUB 110
2015 PRINT"Q INPUT FIELD NAME, FIELD SIZE, FIELD TYPE"
2016 PRINT"     TYPES: 0=BINARY, 1=NUMERIC, 2=ALPHA [CRSR RT]"
2019 RS=0
2020 FOR I=1 TO NF:PRINT"FIELD";I;:INPUTFH$(I),FS%(I),FT%(I):RS=FS%(I)+RS+1:NEXT
2025 A$="I":IF DD=RD THEN A$="I"+STR$(DD)
2030 PRINT#CC,A$
2040 GOSUB 1100:IF E THEN 3900
2050 OPEN4,8,4,STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U,W"
2055 PRINT#4,0;M$;:CLOSE4
2090 GOTO 2120
2100 REM OPEN RANDOM FILE FOR ACCESS
2103 INPUT"QRANDOM FILE NAME";F$
2105 INPUT"KEY FILE DRIVE NUMBER";DD
2110 INPUT"RANDOM FILE DRIVE NUMBER";RD
2120 GOSUB 1200
2140 OPEN4,8,4,STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U"
2142 INPUT#4,RR:IF RR=0 THEN 2147
2145 FOR I=1 TO RR:INPUT#4,K$(I),RR%(I):NEXT
2147 CLOSE4
2150 PRINT"♥Q Q]]]]]]]]]]]]] SAMPLE RANDOM ACCESS Q"
2155 PRINT"TYPE // TO QUIT Q"
2156 PRINT"(HIT RETURN TO ADD RECORD)"
2160 PRINT"QQWHOSE RECORD DO YOU"
2161 PRINT"WISH TO SEE ]]]";RR$
2165 IF RR$=" " THEN 2310
2167 IF RR$="//" THEN 2400
2168 IF RR$="/DIR" THEN GOSUB 4000:GOTO 2160
```

```
2170 FOR II=1 TO RR:IF K$(II)<>RR$ THEN NEXT:GOTO 2300
2175 CR=RR%(II):GOSUB 300
2180 FOR I=1 TO NF:PRINT I;")"FH$(I)":",F$(I):NEXT:PRINT
2190 INPUT "ANY MODS  N";A$:IF LEFT$(A$,1)<>"Y" THEN 2220
2195 INPUT "WHICH FIELD";A
2200 PRINT "  "F$(A):PRINT "";:INPUT F$(A)=VAL(F$(A))
2210 FF=1:GOTO 2190
2220 IF FF=0 THEN 2160
2222 IF A=1 THEN K1$(II)=F$(A)
2225 GOSUB 200
2230 GOTO 2160
2300 PRINT "RECORD NOT PRESENT"
2305 INPUT "DO YOU WISH TO ADD";A$:IF LEFT$(A$,1)<>"Y" THEN 2160
2310 PRINT "**** ADD RECORD ****"
2312 IF FR>NR THEN 2500
2315 CR=FR:FR=FR+1:RR=RR+1
2320 FOR I=1 TO NF:PRINT FH$(I);:INPUT F$(I):F(I)=VAL(F$(I)):NEXT
2330 GOSUB 200
2340 K1$(RR)=F$(1):RR%(RR)=CR
2350 GOTO 2160
2400 REM CLOSE RANDOM FILE
2405 GOSUB 1400
2410 OPEN 4,8,4,"@"+STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U,W"
2420 GOSUB 1000:IF E THEN 3900
2430 PRINT#4,RR;M$
2440 FOR I=1 TO RR:PRINT#4,K1$(I);M$;RR%(I);M$;:NEXT
2445 GOSUB 1000:IF E THEN 3900
2450 CLOSE 4
2455 GOSUB 1000:IF E THEN 3900
2490 END
2500 PRINT "THE FILE IS FULL, NO ADDITIONAL RECORDS MAY BE ADDED"
2510 GOTO 2160
3900 PRINT E,EM$:STOP
4000 FOR DI=0 TO NR:PRINT K1$(DI):NEXT:RETURN
```

Chapter 10

INDEX