commodore COMPUTER

Disk System User Reference Guide

Part Number 320972-01

NOTICE

Commodore makes no express or implied warranties with regard to the information contained herein. The information is made available solely on an "as is" basis, and the entire risk as to its quality and accuracy is with the user. Commodore shall not be liable for any consequential or incidental damages in connection with use of the information contained herein.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

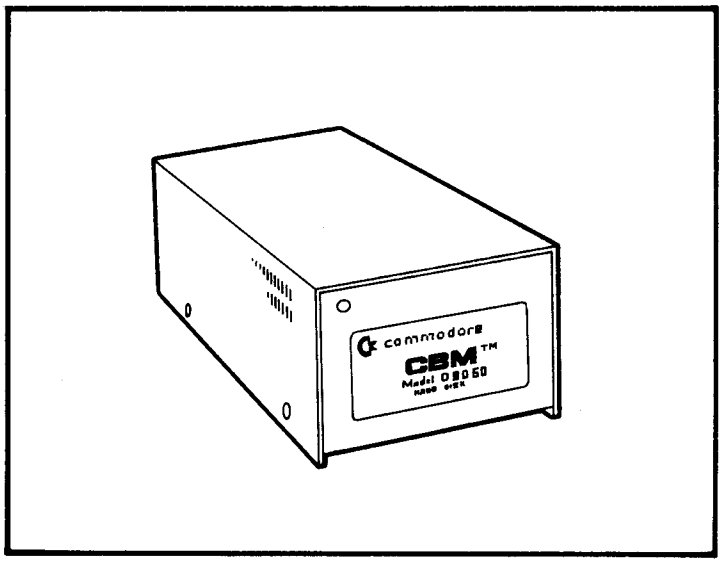
CHAPTER 1. GENERAL INFORMATION

CONTENTS

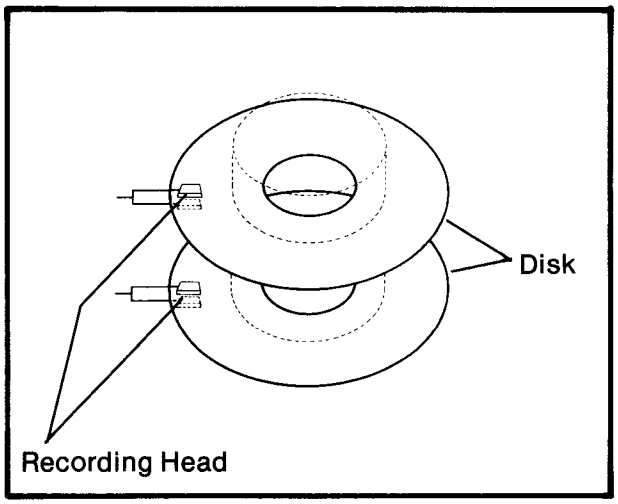
GENERAL DESCRIPTIONS OF COMMODORE DISK SYSTEMS

Description Of The D9060 & D9090

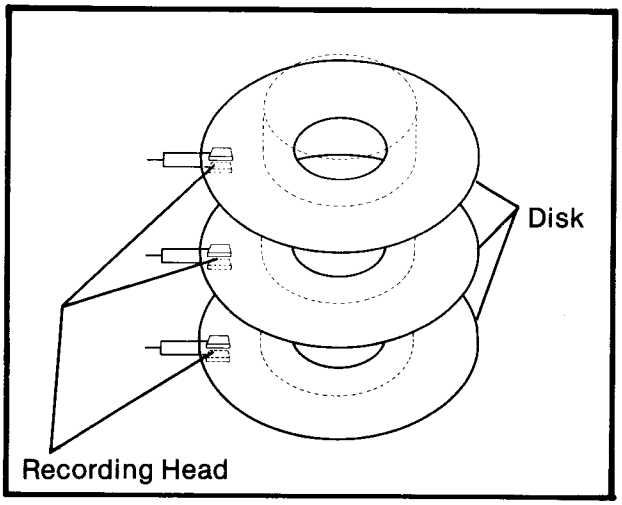
The two models of hard disk units described in this manual are the 5¼" single-drive non-removable "Winchester" technology storage devices. The D9060 and D9090 units feature two or three platters with recording surfaces on both sides and provide respectively 5.0 or 7.5 million characters of storage. A single random access file may occupy the entire capacity of either unit. An IEEE interface connector is located on the back of the drive. Near the lower edge of the rear panel is a "slow blow" fuse, and an AC power cord.



D9060 Hard Disk Unit•Front View
(D9090 Similar)
FIGURE 1.1



D9060 Drive Configuration
FIGURE 1.2



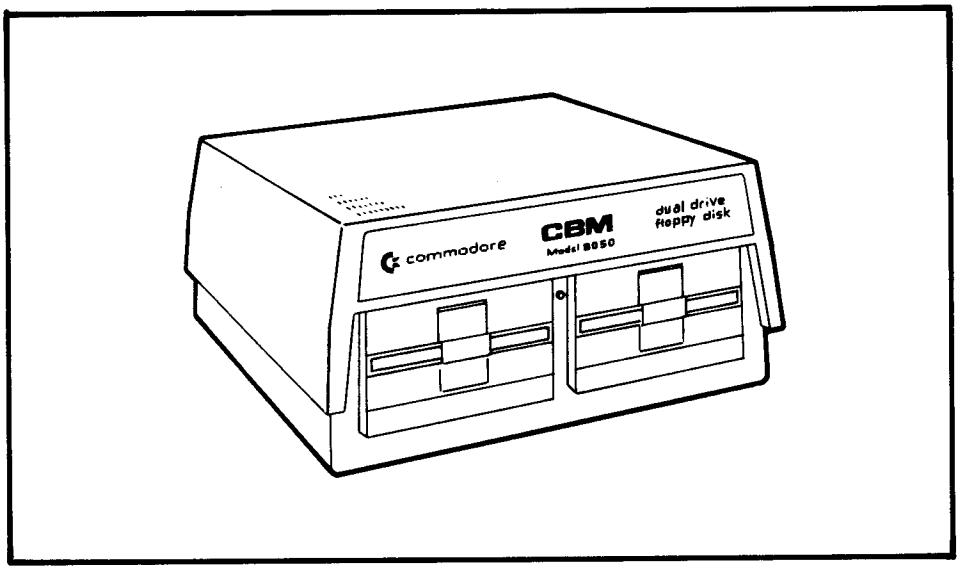
D9090 Drive Configuration
FIGURE 1.3

Description Of The 8250

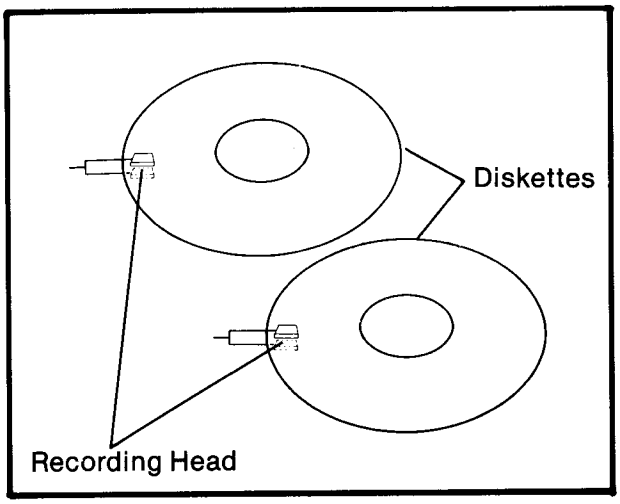
The model 8250 dual floppy disk unit uses a 100 Track Per Inch (TPI) two headed drive with a storage capacity of 1,066,496 bytes (characters) per drive. Each 8250 diskette has 154 tracks, 77 on each side, and is read/write compatible with the 8050 disk drive. A single random access file may occupy an entire 8250 diskette.

Description Of The 8050

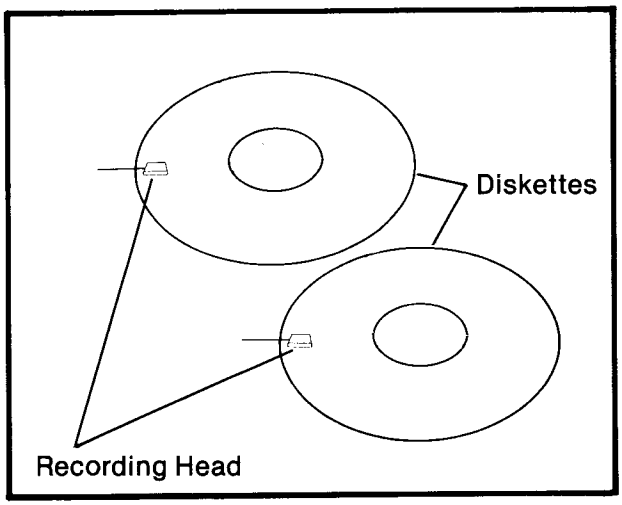
The model 8050 dual floppy disk unit uses a 100 Track Per Inch (TPI) single headed drive with a storage capacity of 533,248 bytes per drive. Each 8050 diskette has 77 tracks, and is read/write compatible with the model 8250 disk drive. This compatibility is limited to one side of the diskette.



8050 Dual Floppy Disk Unit•Front View
(8250 Similar)
FIGURE 1.4



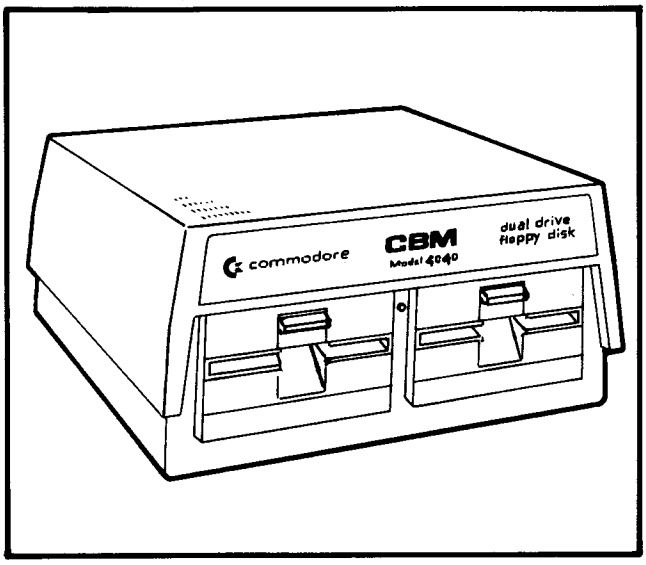
8250 Drive Configuration
FIGURE 1.5



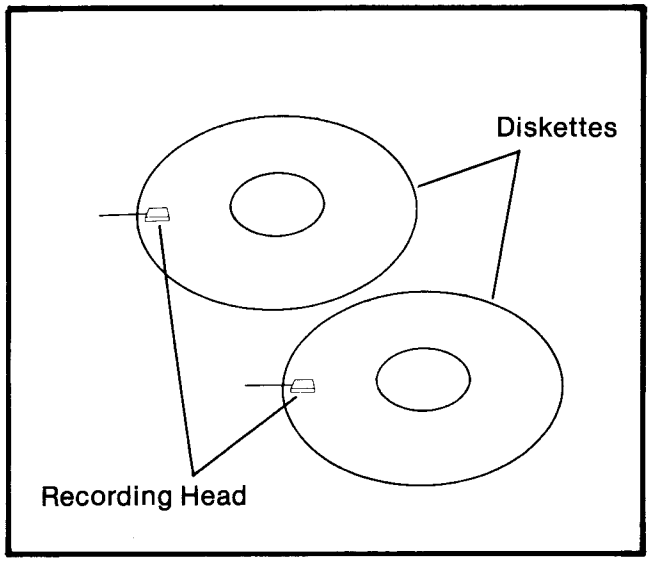
8050 Drive Configuration
FIGURE 1.6

Description Of The 4040

The model 4040 dual floppy disk unit uses 48 Track Per Inch (TPI) single headed drives with storage capacities of 174,848 bytes (characters) per drive. Each 4040 diskette has 35 tracks. Diskettes created on the 4040 drives are read/write compatible with the model 2031 and the VIC-1540 disk units. The 4040 is neither read or write compatible with model 8050 or the 8250 disk units.



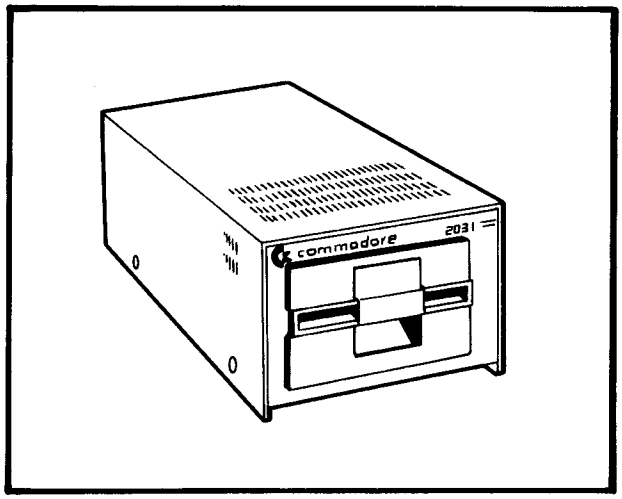
4040 Dual Floppy Disk Unit•Front View
FIGURE 1.7



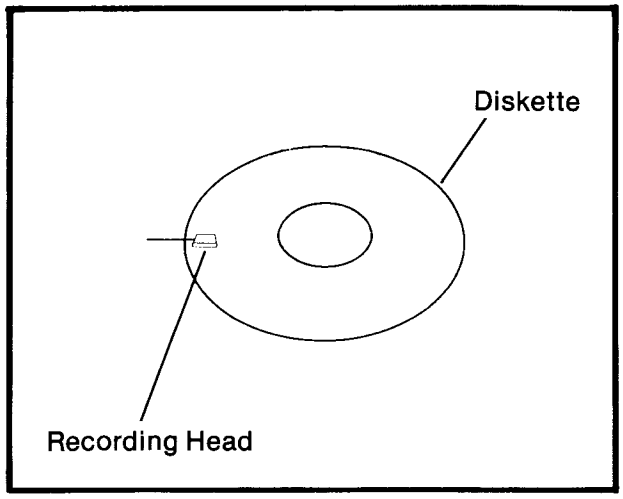
4040 Drive Configuration
FIGURE 1.8

Description Of The 2031

The model 2031 is a low-cost single drive disk unit. The 2031 uses a 35 track (48 TPI) single headed drive with a storage capacity of 174,848 bytes. Diskettes created on the 2031 are read/write compatible with the model 4040 and VIC-1540 disk units. The 2031 is neither read or write compatible with model 8050 or 8250 floppy disk units.



2031 Single Drive Unit
FIGURE 1.9



2031 Drive Configuration
FIGURE 1.10

Unpacking The Disk Unit

A caution: disc units—especially hard disks—are sensitive mechanisms. Be careful not to drop them, either while in their shipping carton or while unpacking.

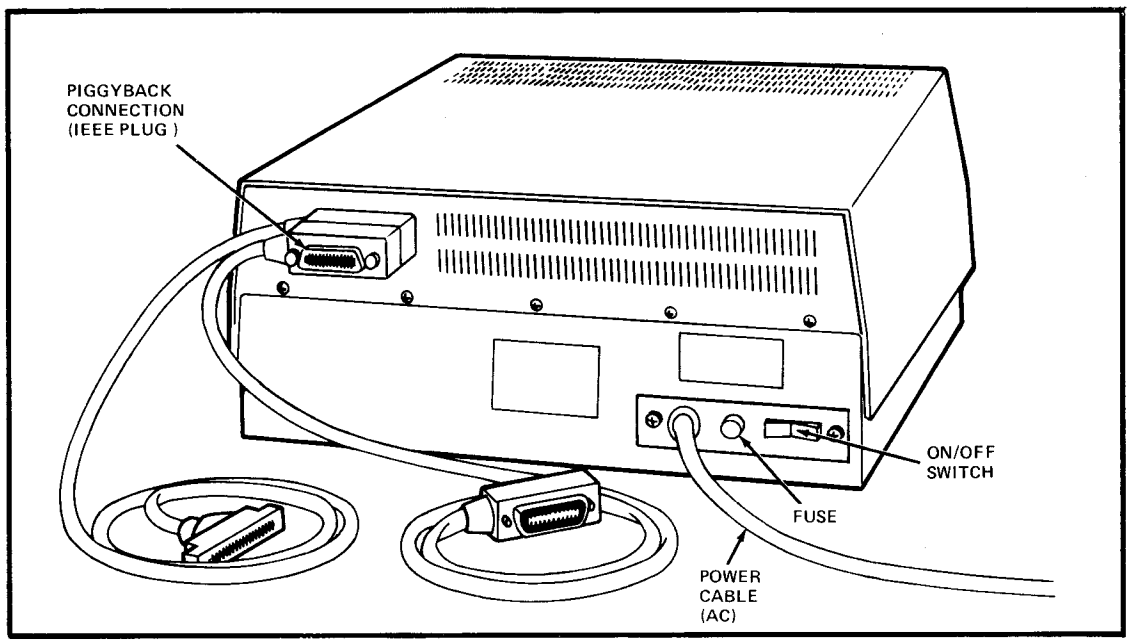
Before unpacking the disk drive, inspect the shipping carton for signs of external damage. If the carton is damaged, caution should be exercised when inspecting its contents. The contents and all packing materials should be removed from the carton. NO packing materials should be discarded until all the contents are located. The carton should contain:

1. Model D9060, D9090, 8250, 8050, 4040, or 2031 disk unit.
2. Commodore Disk Reference Manual.
3. TEST/DEMO Diskette (except for D9060/D9090 hard disk units).

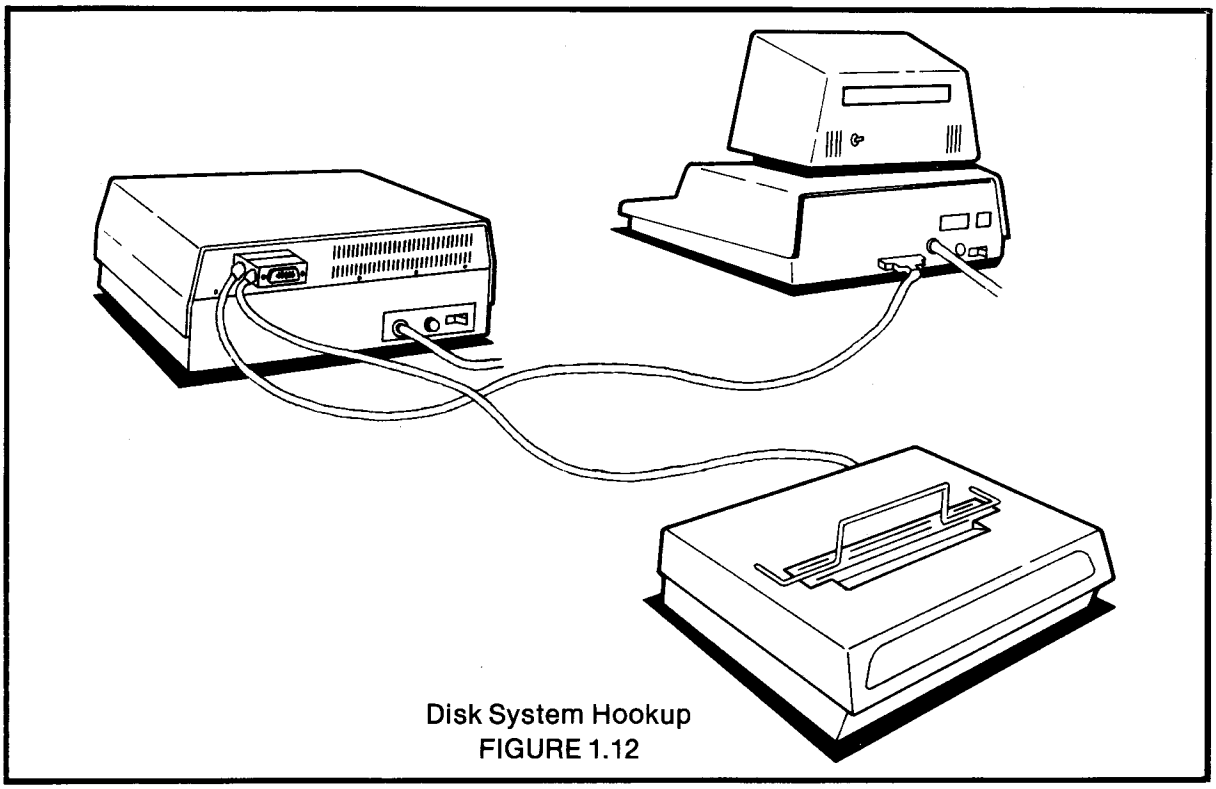
If any item is missing or damaged in shipment, your Commodore dealer should be notified.

Before starting to use the disk drive, make sure it is in good working condition. This includes properly connecting it to the computer, giving it power-on and initial checkout tests, and finally running performance tests using the "PERFORMANCE TEST" program supplied on the TEST/DEMO diskette (except hard disk).

Commodore disk units described in this manual are operationally compatible with any model PET or CBM computer with BASIC 3.0 or BASIC 4.0. VIC-20 and Commodore-64 computers equipped with appropriate PET-IEEE adapter cartridge can also use these disk units.



Models 8250, 8050, 4040 Rear View
FIGURE 1.11



Disk System Hookup
FIGURE 1.12

Connecting The Disk Unit To The Computer

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

PET-to-IEEE cable: Part#4032036-01 (320101)
This cable should be used if the disk drive is the only peripheral to be connected to the computer.

IEEE-to-IEEE cable: Part#4032035-01 (905080)
This cable should be used if the disk drive is to be connected (daisy-chained) to another peripheral device such as the Commodore Model 4022 or other suitably interfaced printer.

NOTE: Be sure to connect the braided-wire ground-strap at *both* ends of the cables.

Procedure for connecting the disk drive to the computer:

Step 1: Power to the computer and all peripherals should be turned OFF.

Step 2: The PET-to-IEEE cable connects between the IEEE-488 interface on the computer and the disk drive. If additional IEEE devices are to be connected, the IEEE-to-IEEE cable(s) must be used.

Step 3: The disk unit power cable should be connected to an AC outlet at this time, but with its power switch turned OFF.

STARTING THE SYSTEM

Read this entire section before preceding.

PERFORMING THE POWER-ON TEST

STEP 1: Power should now be applied to the computer to verify that it is working properly. The following message will be displayed:

```
*** Commodore Basic ***
31743 Bytes Free (Depends on memory size)
ready
```

STEP 2: Power should now be applied to the disk drive. All indicator lights (LEDS) on the front panel should flash twice. The two-color power-on indicator light, on the front of the unit, should stay on. For 4040 and 2031 models, the red LED flashes just once.

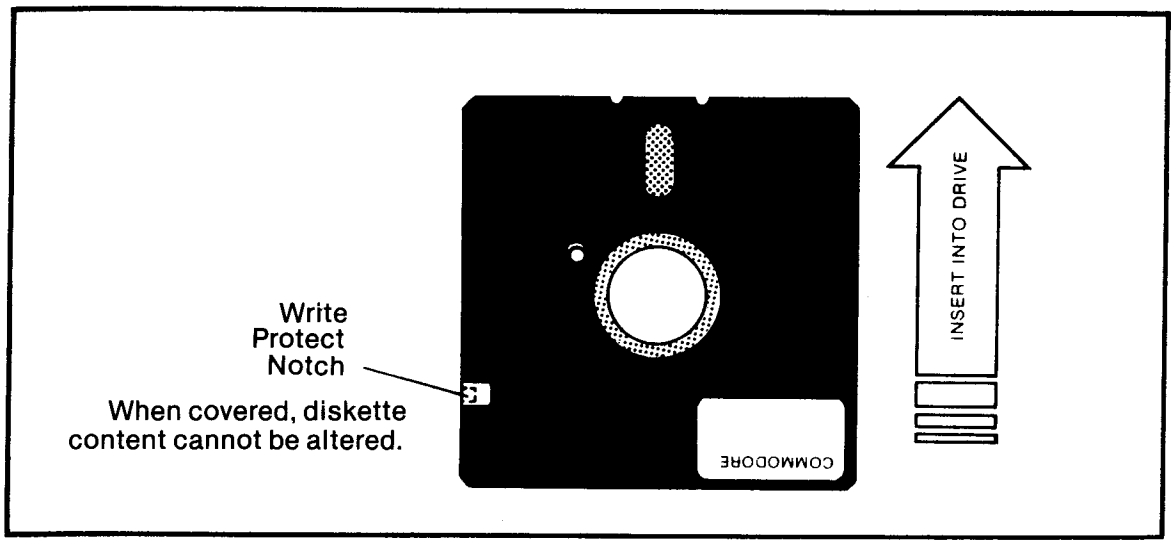
If the drive lights remain on, or all lights flash continously, or if the power/error LED stays red for more than 15 seconds, turn the power OFF. Wait a moment and try again. If these conditions are repeated, all other devices should be removed from the IEEE bus. This will assure that a possible problem related to another device does not affect the disk unit. If the problem persists, your Commodore dealer should be contacted.

NOTE

After applying power to the D9060 or D9090 Hard Disk units, WAIT ONE FULL MINUTE before attempting to use any disk command. This time is required to allow the rotational speed of the disk to stabilize. Any commands issued before this time will cause a DRIVE NOT READY error message to occur, and the drive will not respond to further commands until the INITIALIZE command is used.

LOADING AND CARE OF DISKETTES

As a general precaution, to insure proper seating of diskettes, be sure the diskette is reasonably well centered in its casing before inserting it into the drive. Suggested procedures for inserting diskettes into drives differ from model to model.



Position For Diskette Insertion
FIGURE 1.13

1. 8250/8050: Two types of each of these models exist. One has a unique 'press down' gate which latches the diskette in place and a green LED on each drive. For these drives, simply insert the diskette until it clicks into place; then press the gate latch down firmly, but gently, without hesitation. An automatic motor start-up feature will spin the diskette for a few seconds.

2. The second type of 8250/8050 drive has a 'flip down' door and a red LED on each drive. To properly seat a diskette in these drives, the 'flip down' door should be partially closed once or twice before final closing.

3. 4040/2031: These drives have the same kind of 'flip down' door as the second type of 8250/8050. For proper seating of diskettes the 'flip down' door should be partially closed once or twice before final closing. These drives do not have diskette insertion detect or automatic motor start. Thus, the INITIALIZE command (Chapter 3) should always be used after changing diskettes, before any other command is used. (See Figure 1.7 for 4040 unit. See Figure 1.9 for 2031 unit).

Floppy diskettes are fragile but they can be a long-lasting and very reliable data storage medium when handled properly. Always treat your diskettes gently; never force them into the disk drive. Keep them in their paper sleeves when not in use - in a case designed to hold them. Keep diskettes away from electromagnetic fields, as found near electric motors, power transformers, television sets, or video monitors. Never set heavy objects, such as cups, bottles, or books on top of diskettes. Be prepared for the inevitable unforseen accident: MAKE FREQUENT BACKUP COPIES of your data - and keep the copies in a safe place.

Any 'soft-sector' single-density or double-density certified diskette will work well with Commodore disk units. However, for the 8050 and 8250 disk units, double-density diskettes are recommended. Diskettes with HUB reinforcing rings should not be used with these systems as the automatic centering action of the drive will be adversely affected.

RUNNING PERFORMANCE TESTS

After setting up your floppy disk system and becoming somewhat familiar with its operation you should run the performance test program supplied on the TEST/DEMO diskette to assure that the disk unit is indeed operating correctly. The performance test program exercises the disk system thoroughly under "worst case" conditions of data and timing to assure that the unit will operate reliably under normal use conditions.

To run the performance tests, insert the TEST/DEMO diskette into drive 0 and enter the following commands from the keyboard.

```
dload "performance"          (For computers with BASIC 4.0)
run

load "performance", 8        (For computers with BASIC 3.0)
run
```

The program will display the options and instructions on the video screen for performing individual tests and will display pass/fail results. If the unit fails on any test your Commodore dealer should be notified.

CHAPTER 2. FUNDAMENTALS OF USING COMMODORE DISK SYSTEMS

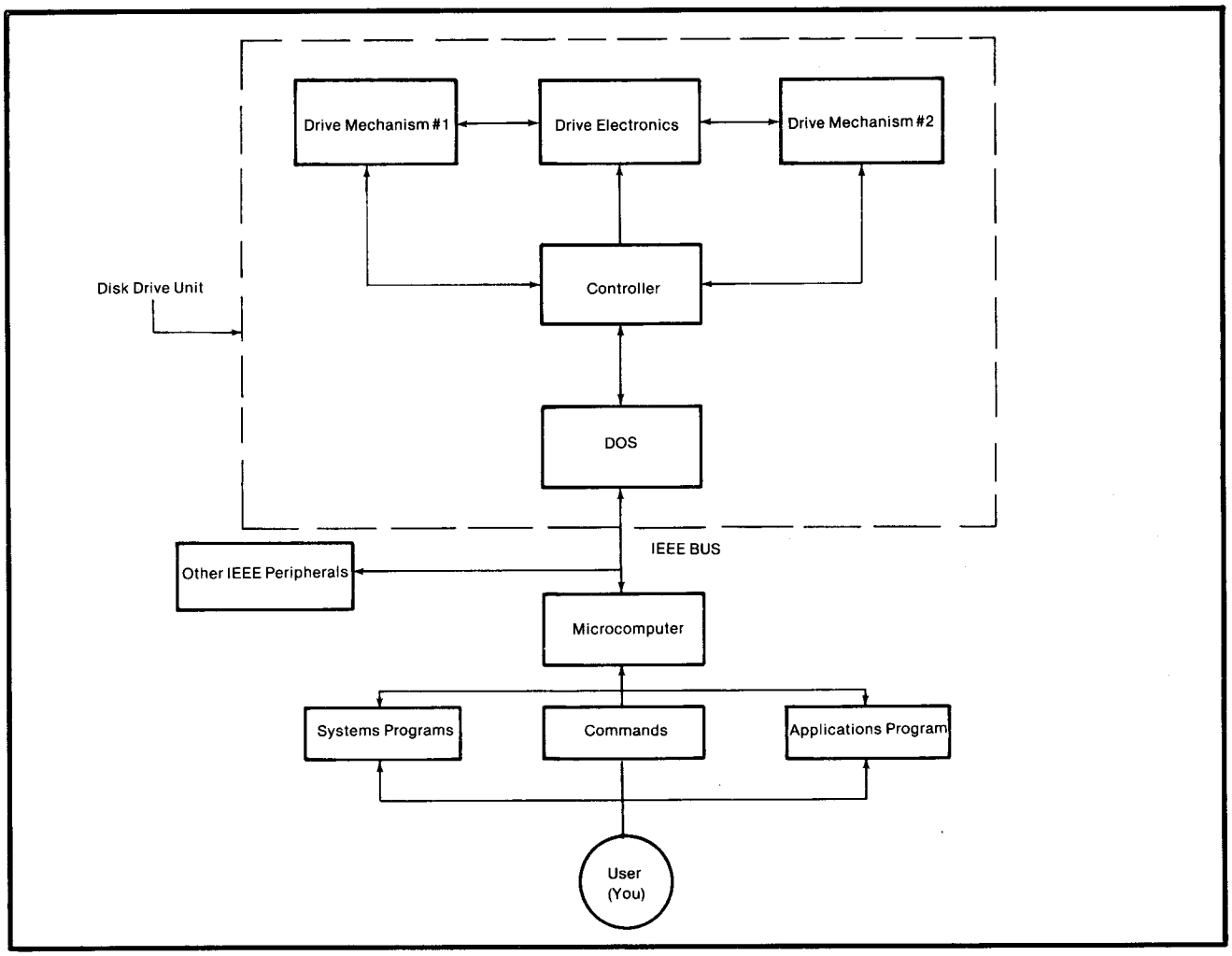
CONTENTS

CHAPTER 2. FUNDAMENTALS OF USING COMMODORE DISK SYSTEMS

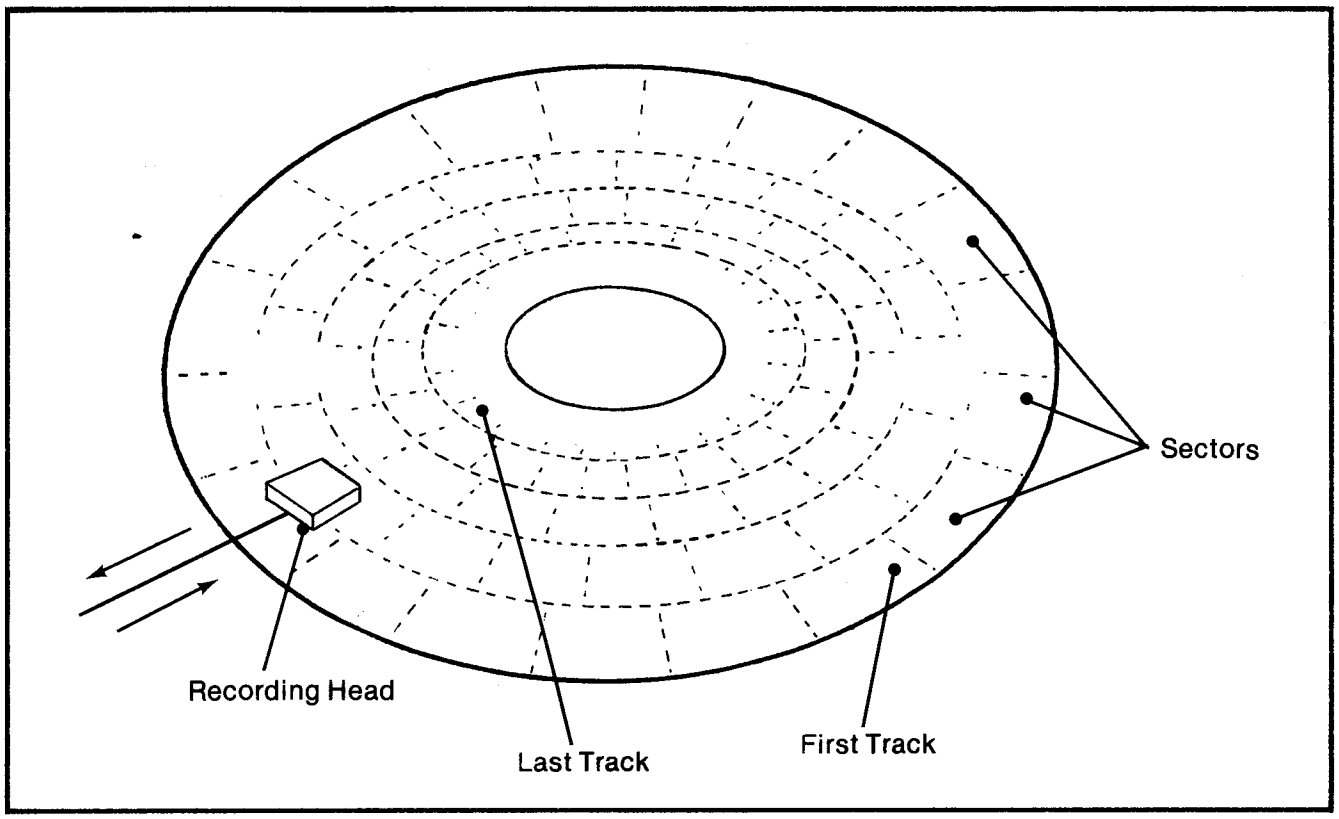
It is best to think of your computer and disk units as parts of an overall system. An understanding of the concept that each individual component of the system acts and reacts to commands and signals from other devices in the system will greatly accelerate your grasp of how to operate, control, and master the system.

Commodore disk systems are designed around two central concepts: using intelligent peripherals to relieve the computer (and the programmer) of most of the detailed tasks involved in data management; and providing large file-handling capabilities supported by flexible BASIC programming statements and a comprehensive Disk Operating System command set.

All Commodore disk units are "intelligent" peripherals, containing their own microprocessors, read-only and read/write memory. In fact, the Disk Operating System program (DOS) resides in the disk unit. Data transfers are initiated by means of commands sent from the computer to the disk unit and DOS handles the tasks of locating, storing, and retrieving data from the disk and sending it to the computer.



Disk System Diagram
FIGURE 2.1



Commodore Disk and Recording Head
FIGURE 2.2

DISK STORAGE

You now have a device which can store thousands or even millions of characters of data. These characters can be program instructions, text for letters, the inventory data for your company, or almost anything. The disk stores this information by means of changing magnetic patterns on the disk surface. This is the same method used to store information on magnetic tapes.

With a tape all data must be read from the beginning of the tape to find your information. This is called sequential access. A disk drive can go directly to any position on the disk and read the data. This is called random or direct access. Commodore disks may also use sequential access, but they can start at any place on the disk to begin reading data.

Data is stored on disk in physical locations called tracks. Each of these tracks is further divided into sectors. Each sector can store up to 254 bytes (characters) of data. Figure 2.2 illustrates the physical storage of data on Commodore disks. A file uses as many of these sectors as needed to hold its information.

DISK FILES

Your information is stored "physically" in tracks and sectors. It is stored "logically" in files. A file is an ordered collection of characters. The contents of the file are of no concern to the disk drive, only to you. Chapter 3 explains the commands that allow you to work with your files as logical items of information. You may further divide your files into records or variables, and that is discussed in Chapter 4.

If you had to maintain information of where all your files were kept it would consume most of your programming time. Commodore disk units are always executing a program that takes care of the locating, storing and reading of the actual tracks and sectors you access. This program is called the "DOS", or Disk Operating System. DOS allocates storage as needed, finds your data for you, and keeps you from having to be concerned with the details of managing your information.

THE DISK OPERATING SYSTEM (DOS)

Inside the disk unit is a computer executing a program called DOS. DOS is controlling the disk drive hardware. When you want information, a command entered on the keyboard or used in a program is sent to DOS. The command is interpreted and the necessary operations are performed to supply you, or your program, with the desired information. This built-in "intelligence" of Commodore disk systems relieves the computer and the programmer of disk storage management and provides a high degree of compatibility with the various models of Commodore computers.

Commodore BASIC can maintain ten open files at once. DOS allows access to a maximum of five disk files at once. BASIC can, therefore, access five disk files and five files on other devices, such as printers, modems, plotters, etc., from within a program.

DOS uses part of the disk for storage of information about the disk. It keeps a list of the file names on the disk in a section called the Directory. It also keeps a list of the allocated sectors and the free sectors in the Block Availability Map (BAM).

THE BLOCK AVAILABILITY MAP (BAM)

The BAM is a "map" representing both available and allocated (used) sectors (or blocks) on a disk. Formatting a disk creates the BAM on the disk. During subsequent disk operations the BAM is read from disk into DOS memory within the disk unit. The BAM is stored in various locations on the disk depending on the model of the disk unit.

When the system stores data on the disk, the BAM will be referenced by DOS to determine whether space is available. For Sequential or Program files DOS checks for space before each block of the file is written. If a free block is found, the BAM is updated to account for the space used and the data block is written to the disk. If no free blocks are available an error message will be generated.

As changes occur to the BAM in DOS memory, the BAM on disk will be updated periodically to reflect these changes. Updates to the BAM occur when a program is stored on disk with the DSAVE command, or when DCLOSE is performed on a Relative or Sequential data file. One block of the BAM is loaded into DOS memory at a time. When updated, this block is written back to the disk and (on some models) another block is loaded into DOS memory. This interchange of information between the two BAMs, one in DOS memory and the other on disk, enables the system to maintain a record of free and allocated space on the disk.

COMMUNICATING WITH DOS

Transfer of data to peripherals, such as printers, cassettes, disk drives, and other external devices can sometimes become complex. The computer must be told to open a path, or "channel", for the data transfer. This is done by the OPEN or DOPEN# statements in CBM BASIC. The DOPEN# statement or command is used to access disk drives. After the data is transferred you tell the computer to stop using the channel, and free it for other use, by giving the CLOSE or DCLOSE# statement. The CLOSE and DCLOSE# statements also update the BAM.

Channel numbers can range from 0 thru 255 and do not have permanent assignments, but are assigned arbitrarily by the programmer. Channel numbers are referred to through this manual as "logical file" numbers. The logical file number relates DOPEN#, DCLOSE#, INPUT#, GET#, RECORD# and PRINT# statements with each other and associates them with the file name and physical device being accessed.

Each device attached to the computer has its own physical device (or unit) number (8 thru 15 for disk units) to which it responds when being accessed by the computer. The device number is used as a command parameter when opening a file to identify the physical unit to be accessed.

CBM disk units are preset at the factory as physical device number 8. The number may be changed in the unit as described in Appendix C, or it may be temporarily altered using a DOS command as described in chapter 6. If a device number is not specified in a command, DOS assumes unit number 8.

In addition to a logical file number and a device number, Commodore disk units also respond to several "secondary addresses". These are best visualized as "function codes" from the computer telling the disk unit what operation to perform.

Address 0 is used (with DLOAD) to read a program file into the computer memory. Address 1 is used (with DSAVE) to store programs into disk files. Addresses 2 to 14 are used to access data files. Address 15 is a special "command channel" address used with the OPEN statement to perform many of the special disk operations described in this manual and to retrieve status information about the disk operations.

FILE NAME PATTERN MATCHING

Pattern matching of file names is available on all Commodore disk units. Pattern matching is the use of question marks (?) and asterisks (*) with the ‹file name› parameter on disk commands. Using pattern matching causes DOS to perform an operation on several files with similar names using a single command only once.

The asterisk is used at the end of a string of characters to indicate that the rest of the name is to be ignored in the search for matching file names. For example "FIL*" could refer to files named:

FIL
or FILE 1
or FILEDATA
or FILLER
or Any other file name starting with the letters FIL.

The question mark may be used anywhere within a string of characters to indicate that only the characters 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.FIL

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

The question mark and the asterisk may be combined in many useful ways. However, the asterisk should always appear as the last character in any pattern, whether or not question marks are used. For example the pattern "J*????" does not make sense because the question marks are in an area which is insignificant because of the asterisk.

The pattern "P???FIL*" will access files with the names:

PET FILE
or PRG-FILE-32
or POKEFILES$$
or Any other file starting with "P" and having "FIL" in positions 5 thru 7.

DLOAD "*" will load the first file (which must be a program file) in the disk directory on drive 0. DOPEN# with pattern matching may be used to open an existing file, in which case the first file that matches the pattern will be opened. DOPEN# must not be used with pattern matching when creating a new file.

The SCRATCH command with pattern matching should be used carefully, since multiple files will be scratched. Never use RENAME or DSAVE with pattern matching or an error will result. COPY may be used with pattern matching for the source file names, but the only pattern permitted for the destination file is the special case "*".

COMMAND ABBREVIATIONS

Whether entered directly from the keyboard or included in a program, DOS commands many appear with their full spelling or in abbreviated form. Commands are abbreviated by entering enough characters of the command name to uniquely distinguish it from any other DOS command or BASIC keyword. All but the last character of the abbreviation are keyed unshifted and the last character shifted.

For example, "catalog" and "cA" are identical to DOS, as are "print#1" and "pR1". Abbreviation of commands does not reduce memory usage in programs, but is supported as a convenience for users of the system. When a program containing command abbreviations is listed, the commands will appear in fully spelled form.

USING PROGRAM VARIABLES WITH COMMANDS

Each disk command or statement has associated with it one or more optional parameters which may be used to specify file names, drive numbers, etc. When needed, command parameters may appear in either of two forms. Parameters may be stated explicitly, such as: DOPEN#1,d1,"Inventory File" or BASIC variable names enclosed in parentheses may be used, such as: DOPEN#1,(dn),(A$). The two DOPEN# commands would produce the same results.

When entering disk commands from the keyboard, parameters must be stated explicitly. When used in programs, parameters may be either explicit or variable, and both forms may be used in the same command.

CHAPTER 3. DIRECT DOS COMMANDS

CONTENTS

CHAPTER 3. DIRECT DOS COMMANDS

CONVENTIONS USED TO DESCRIBE COMMANDS

Throughout this manual certain conventions are used to describe the syntax of disk commands and to show both required and optional parts of commands. The rules for interpreting command syntax are as follows:

1. Command names and other required words are shown in capital letters. They must appear where shown in the command, entered as unshifted letters, spelled exactly as shown.

2. Items shown within quotation marks (" ") indicate variable data which must be supplied by the user. Both the quotation marks and the data they enclose must appear in the command.

3. Items enclosed in square brackets ([ ]) indicate an optional command parameter which, if present, contains data supplied by the user.

4. Items underlined within square brackets indicate required characters used in optional parameters. If the optional parameter is used, the required characters must be entered as unshifted letters spelled exactly as shown.

5. Items within angle brackets (‹ ›) indicate variable data which must be supplied by the user.

The syntax shown for each command uses BASIC 4.0 format. Appendix A shows how the earlier versions of BASIC accomplish the same task.

Example of syntax format:

HEADER" ‹name› "[,d ‹drive›][onu ‹unit›][,i ‹id›]

Example of entered command:

HEADER "Inventory",d1onu9,i05

In actual use, the sequence of parameters appearing in many commands may differ from that shown in the syntax examples in this manual. The examples are not meant to show all possible sequences but are intended to present all required or optional parameters.

The following table shows examples and descriptions of the symbols used for various command parameters in the following chapters. It is not meant to show every possibility, but to give you a better insight into the syntax that follows.

SYMBOL	EXAMPLE	DESCRIPTION
‹ drive ›	0	A physical drive number
‹ unit ›	8	A hardware device number
"‹ file name ›"	"Inventory"	An actual file name
"‹ name ›"	"old data"	Any file or disk name
‹ dest drive ›	1	The drive number for the destination during a transfer of data.
‹ src drive ›	0	The drive number for the source of data during a transfer.
‹ address ›	15	A secondary address "Function code".
‹ file num ›	67	A logical file number or channel number.
‹ rec len ›	128	Physical record length used for relative file access.
‹ var name ›	A$	A program variable name.

The DOS commands described in this chapter are called "direct" commands because they are not intrinsically a part of the CBM BASIC language. These commands are usually entered via the keyboard for immediate action, though they may also appear as statements in BASIC programs.

The same is true for the statements described in Chapter 4 which usually appear in BASIC programs, but may also be entered via the keyboard. In general, the terms "command" and "statement" are used interchangeably in this manual.

NOTE: The drive number reference in all DOS commands has been maintained in the examples which follow to be compatible with dual disk drive units. If using the 2031 single disk or the D9060/D9090 hard disk drives, all references to the drive number must be a zero (0). Any reference to drive 1 will result in an error condition.

DISK LEVEL COMMANDS

HEADER Used to format a disk

The HEADER command prepares a new disk for its first use. The HEADER command must be performed (only once) on any new disk before it can be used for data or program storage.

The HEADER command writes track and sector addresses on the disk, writes binary zeroes in all data blocks, and creates the BAM, Directory Header, and the Directory for the disk. This process is sometimes called "formatting" a disk. Any data that may exist on the disk is destroyed during this process.

Because of the irreversible nature of this command, DOS asks "Are You Sure?" before proceeding. Typing a "y" in response to the question allows DOS to proceed. Any other response causes the command to abort without writing on the disk.

Syntax — HEADER "‹ name ›"[,d‹ drive ›][onu‹ unit ›][,i‹ id ›]

Example:

```
HEADER "Inventory",d0onu9,i04
Are You Sure? y
```

This formats the disk in drive 0, unit number 9, giving it a name of "Inventory", and an ID of 04. If the drive number is omitted DOS assumes drive 0. The disk ID may be any two characters selected arbitrarily and the disk name may be up to 16 characters of any type.

OPTIONAL SYNTAX

For previously used disks DOS provides for an optional syntax. If the disk ID parameter is omitted, DOS will re-write the Directory Header, and re-write an empty BAM and Directory. This can be considered equivalent to a "SCRATCH" for all files on the disk, but requiring less time to execute.

Syntax — HEADER "‹ name ›" [,d‹ drive ›][onu‹ unit ›]

Example:

```
HEADER "parts"
Are You Sure? y
```

The name of the disk in drive 0 is changed to "parts". The disk ID is not changed. All data on the disk is lost, because the Directory is erased and all data blocks are made available for use.

INITIALIZE Log a Change of Diskette

When a diskette is inserted into a drive, for any reason, that drive MUST be initialized to ensure that the BAM in DOS memory is the proper data for the diskette currently in the drive. Failure to properly initialize the drive may cause a DISK ID MISMATCH error, or loss of data. Most models of disk units handle initialization automatically.

The 4040 and 2031 disk units check diskette ID each time a drive is addressed to find whether initialization is needed. If a new diskette ID is detected the drive is initialized without need for operator action. If the ID is the SAME as the previous diskette a change of diskettes WILL NOT be detected and data will be lost if the INITIALIZE command is not used before any other command. Thus care should be taken to assign unique ID codes for 4040 and 2031 diskettes.

Since the 8050 and 8250 disk units feature automatic detection of diskette removal/insertion, these units will self initialize when the drive door is closed (8050 'gate' type) or when the drive is first addressed (8250 and 8050 'door' type). Hard disk units self initialize during power-up and need no further attention.

The normal BASIC 4.0 syntax is not used for the INITIALIZE command. The following command sequence must be used.

Syntax — OPEN ‹ file number ›,‹ unit ›,‹ address ›
PRINT#‹ file number ›,"I[NITIALIZE]‹ drive ›"
CLOSE‹ file number ›

Example:

```
OPEN 5,8,15
PRINT#5,"initialize0"
CLOSE 5
```

This initializes drive 0 of disk unit 8. The secondary address 15 in the OPEN statement is the "command channel" which must be used whenever PRINT# is used to transmit a command to a disk unit. If the drive number is omitted from the PRINT# statement both drives will be initialized on dual drive units. The word "initialize" may be abbreviated to "I".

DIRECTORY/CATALOG Display Disk Directory contents

The DIRECTORY command displays the list of files stored on a disk onto the screen. If a drive reference is omitted, the directories of both disk drives are displayed on dual drive systems. While the directory is listing the following business keyboard entries affect the display:

1. The RUN/STOP key aborts the listing.
2. The colon (:) halts the display temporarily until the 9, 6, or 3 key is pressed. (Business only)
3. The 9, 6, or 3 key resumes display.
4. The left arrow key (←) causes a slow display.
5. Holding the colon and pressing RUN/STOP lists one line at a time.

For graphics keyboards the space bar halts the display until the space bar is pressed again. The RUN/STOP key aborts the display.

Syntax — DIRECTORY [d‹ drive number ›][onu‹ unit ›]
or CATALOG [d‹ drive number ›][onu‹ unit ›]

Example:

DIRECTORY d1 or CATALOG d0onu9

A sample directory printout shows the following:

```
0 "8050 DEMO    " 8D 2C
5   "UNIVERSAL WEDGE"  PRG
8   "UNIT TO UNIT"     PRG
3   "CHANGE 8050"      PRG
27  "PRINTER DEMO"     PRG
12  "SEQUENTIAL"       PRG
11  "PERFORMANCE TEST" PRG
5   "CHECK DISK"       PRG
17  "LOGIC DIAGNOSTIC" PRG
1964 BLOCKS FREE.
```

The above display shows the following information about the disk:

On first line: 0 "8050 DEMO " 8D 2C

drive number — volume name — disk ID — DOS version

On other lines: 5 "UNIVERSAL WEDGE" PRG

blocks used — file name — file type

File types are: PRG program file REL relative file
SEQ sequential file USR user file

At the end of the directory display is the quantity of free blocks remaining on the disk.

PRINTING THE DISK DIRECTORY

The directory display can be sent to a printer with the following command sequence:

OPEN1,4	Opens a channel to device 4 (printer)
CMD1	Changes the screen output to device 4
DIRECTORY d0	Prints the directory of drive 0
PRINT#1	Returns output to the screen
CLOSE1	Closes printer channel

By replacing the OPEN statement with DOPEN#1,d0,"files", a sequential data file on drive 0 called "files" would receive the contents of the directory display. A program could then read the directory information from "files".

COLLECT Rebuild BAM and delete open files

The COLLECT command is used to re-create a valid BAM on a disk when files have been left "open" because of programming errors. COLLECT traces through an entire disk and rebuilds the BAM from the disk data contents. If COLLECT encounters an open file in the directory, that file is deleted and the blocks used by it are freed for other use. Any User-type files on the disk are also deleted.

If a disk read error occurs during the COLLECT command execution, the process is aborted and the disk remains in its original state.

Syntax — COLLECT [d‹ drive ›][onu‹ unit ›]

Example:

COLLECT d0

Rebuilds the BAM on the disk in drive 0, and deletes any open files or User-type files.

BACKUP Duplicate an entire disk

The BACKUP command creates an identical copy of the entire disk, including the BAM, the disk name, the disk ID, the directory and all files. The drive number references are required.

Syntax — BACKUP d‹ src drive ›[onu‹ unit ›] TO d‹ dest drive ›[onu‹ unit ›]

Example:

BACKUP d1 TO d0onu9

This copies the entire contents of the disk in drive 1 on unit 8 to drive 0 on unit 9.

Because various models of disk systems differ in storage capacity and the organization of data storage, the BACKUP command may not be used between disk units of different models. The only exception is the 2031 and 4040 models which use identical diskette formats.

FILE LEVEL COMMANDS

DSAVE Save a program to disk

This command transfers the program in computer memory to the disk unit, and places it into the file ‹ file name ›. If the file already exists an error condition results and the file is not stored. The file will be type "prg". If the first character after the opening quote mark in the file-name is an 'at sign' (@) then if a file of the same name exists it will be replaced by the program in memory.

Syntax — DSAVE [d‹ drive ›][onu‹ unit ›],"[@]‹ file name ›"

Example:

DSAVE d1,"payroll"

This stores the program in computer memory into the file "payroll" on drive 1. The drive parameter defaults to zero if not present.

Example:

(DSAVE "@TEST.prg", d0)

This replaces the file TEST.prg on drive 0 with a newer version of the program. If no such file-name exists, it is created.

DLOAD Load a program from disk

The DLOAD command transfers a program from disk into the computer memory. DLOAD also closes all open data files and the disk command channel. The drive number defaults to 0 if not specified.

Syntax — DLOAD [d‹ drive ›][onu‹ unit ›],"‹ file name ›"

Example:

DLOAD d0,"mainmenu"

Load the program "mainmenu" from drive 0, unit 8, into computer memory.

SPECIAL SYNTAX

The command LOAD "*",8 causes the first file in the directory of drive 0 of unit 8 to be loaded. This must be a program file or the error FILE NOT FOUND is generated.

AUTO START FEATURE

DOS provides for automatic start-up of a program by holding the SHIFT key and pressing the RUN/STOP key on computers with 4.0 BASIC.

This causes the first file on the disk directory of drive 0 on unit 8 to be loaded and execution to begin on the first line of the program. If the first file in the directory is not a program file the error FILE NOT FOUND is generated. A menu program could be used with this feature.

RENAME Rename a file

The RENAME command changes the name of a file in the disk directory. The data in the file is not altered. If a drive reference is not given this command searches both drives on a dual drive system. Pattern matching of file names is not permitted. The new name for the file must not exist on the disk where the file resides. Duplicate names are not permitted or created.

Syntax — RENAME [d‹ drive ›][onu‹ unit ›],"‹ original name ›" TO "‹ new name ›"

Example:

RENAME d0, "history" TO "older"

This renames the file on drive 0, unit 8, from "history" to its new name of "older". Data in the file is unchanged.

COPY Copy one or more files

The COPY command duplicates the contents of selected program or sequential files. Files may be copied to the same or different drive or unit of any model. Source and destination file names may be the same if the source and destination drive or unit number is different.

Pattern matching of source file names is permitted. The only pattern permitted for the destination file names is "*" which causes the source files to be copied without changing their names. If the source and destination drive/unit are the same, the file names must be different.

Syntax —

COPY [d‹ src drive ›][onu‹ unit ›],"‹ src name ›" TO [d‹ dest drive ›][onu‹ unit ›],"‹ dest name ›"

Example:

COPY d0,"records" TO d1onu9,"backups"

This duplicates the file "records" from drive 0 into a file called "backups" on drive 1, unit number 9.

If the source drive reference number is omitted, both drives will be searched for the files to copy.

Example:

COPY d0,"???basic*" TO d1,"*"

This uses pattern matching to copy all program and sequential files on drive 0, having the letters "basic" in positions 4 thru 8 of their file names, to drive 1. The destination file names on drive 1 will be identical to the actual source file names.

CONCAT Append one file to another

The CONCAT command concatenates or appends the source file to the end of the destination file. Both file names must exist or an error is generated. The contents of the source file are unaltered. Pattern matching of the file names is not permitted in this comand.

This command can only be used with sequential ("seq") file types.

Syntax —

CONCAT [d‹ src drive ›][onu‹ unit ›],"‹ src file ›" TO [d‹ dest drive ›][onu‹ unit ›],"‹ dest file ›"

Example:

```
CONCAT d1,"april" TO d1,"yearly"
```

This example adds the contents of the sequential file "april", on drive 1 to the end of the sequential file "yearly" on drive 1. The "april" file is unchanged.

SCRATCH Delete a file or files

This command destroys unwanted files. The file names are removed from the disk directory and their data space is freed for other use. Pattern matching is permitted in this command, but should be used with caution.

As a built in safety feature, this command asks for verfication in the same way as the HEADER command. Entering a "y" in response to the "Are You Sure?" question allows the SCRATCH to continue, any other response aborts the command.

Syntax — SCRATCH [d‹ drive ›][onu‹ unit ›],"‹ file name ›"

Examples:

```
SCRATCH d1,"temp"
Are You Sure? y
```

Deletes the file named "temp" on drive 1, unit 8.

```
SCRATCH "*"
Are You Sure? y
```

Deletes every file presently on the disk in drive 0.

```
SCRATCH d0,"basic*"
```

Deletes every file on drive 0 with a file name whose first five characters are "basic".

CHAPTER 4. USING DOS FROM BASIC

CONTENTS

CHAPTER 4: USING DOS FROM BASIC

This chapter describes the commands that are used to access the data in a file. The commands may be directly executed or they may be included in programs as statements.

The data file types referenced in this chapter are either "seq" for sequential, or "rel" for relative. The command syntax conventions are the same as Chapter 3.

DOPEN# Prepare a File for Access

SEQUENTIAL FILES

This command tells DOS to prepare a sequential type file for access.

When the file is opened for reading the ‹file name› specified must exist or an error is generated. If the file is already open an error is also generated. If the file is open for writing the ‹ file name › must not currently exist, and the file will be created.

Syntax —

TO READ — DOPEN#‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›",R
TO WRITE — DOPEN#‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›",W

Example:

```
DOPEN#1,d1,"test records",R
```

Open the file "test records" on drive 1, as logical file number 1 for reading.

```
DOPEN#2,d1,"print file",W
```

Create the file "print file" on drive 1 for write operations.

RELATIVE FILES

Relative files allow access to any record in the file in any order. The RECORD# (see page 37) command is then used to specify which record in the file to access.

Opening a relative file for access allows both reading and writing. If the file does not exist it will be created. If the file is already open with the same ‹file number› an error is generated. The record length ‹rec len› parameter is optional when opening a file which already exists. Relative files are discussed in greater detail in Chapter 5.

Syntax — DOPEN#‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"[,L‹ rec len ›]

Example:

```
DOPEN#5,"Payables",l80
```

This prepares logical file number 5, named "Payables" on drive 0 for read and write access. The ‹ rec len › specified is checked (if "Payables" already exists) against the actual record length on disk and if there is not a match an error is generated.

The number of files which DOS can allow to be open concurrently depends on the mix of sequential and relative file types opened. The following maximum combinations are permitted on all models except the 2031.

0 relative and 5 sequential
or 1 relative and 3 sequential
or 2 relative and 2 sequential
or 3 relative and 0 sequential

On the 2031 the combinations of open files permitted are:

0 relative and 2 sequential
or 1 relative and 1 sequential

APPEND# Continue a Sequential File

The APPEND# command opens a sequential file to permit more data to be written to the end of the file using the PRINT# statement. The file must currently exist or an error message is generated.

Syntax — APPEND#‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"

Example:

```
APPEND#3,d0,"translog"
```

The file named "translog" on drive 0 is opened for writing and the DOS file access pointer is positioned to the character position following the current end of the file. If the drive number is not specified DOS assumes drive 0.

DCLOSE# Quit file processing

The DCLOSE# command closes files opened with the DOPEN# command. Specific files can be closed by supplying a ‹ file number › reference, or all files can be closed by omitting the number. The DLOAD command also closes all files.

This command causes the BAM to be updated with any information still in DOS memory, and updates the directory with the current file size. Forgetting to DCLOSE# a file before changing disks will cause data loss.

Syntax — DCLOSE#‹ file number › close one file
DCLOSE close all open files

Example:

```
DCLOSE#36
```

This closes logical file number 36.

BASIC can maintain a maximum of ten open files. These may include up to five disk files and five files on other devices such as printers or modems.

PRINT# Write data into a file

The PRINT# command is used to send data to a file. The file must have been opened for writing with the DOPEN# command. The ‹ file number › must be the same as the logical file number assigned with the DOPEN# command.

Syntax — PRINT#‹ file number ›[,‹ variable ›][,‹ variable ›]...[,‹ variable ›]

Example: PRINT#5,A$

This transmits the contents of the ‹ variable › named A$ to logical file number 5. The ‹ file number › was associated with a file name when the file was opened.

The punctuation characters which may be used to separate variable names in the output list are: blank, comma, and semi-colon. Punctuation characters may be omitted between string type variables or after the last variable in the statement. A comma must appear before the first variable (if any) in the output list.

Whether blanks, commas, semi-colons, or no punctuation characters are used between variable names, the effect is the same. The contents of variables in the list are concatenated into a single string followed by a carriage return character (CHR$(13)).

When using BASIC 4.0 any file opened with a logical file number greater than 127 will cause a line-feed character (CHR$(10)) to be transmitted to the file after the carriage return at the end of an output variable list. Logical file numbers less than 128 will not send the line-feed character. With earlier versions of BASIC the line-feed is always sent, regardless of logical file number, unless steps are taken to suppress it.

Examples: where variables A$= "AAA" and B$= "BBB" and C$= "CCC".

```
PRINT#2,A$;B$;C$
PRINT#3,A$,B$,C$
PRINT#4,A$ B$ C$
```

All send the data: AAABBBCCC‹ carriage return ›

```
PRINT#150,A$,B$,C$
```

Sends the data: AAABBBCCC‹carriage return› ‹line-feed› when using BASIC 4.0 The ‹file number› of 150 causes the line-feed character.

```
PRINT#5,A$;B$;C$;CHR$(13);
```

Sends the data: AAABBBCCC‹ carriage return › when using any version of BASIC. The trailing semi-colon will suppress the output of the carriage return and/or line-feed characters which would normally be sent.

INPUT# Read data from a file

The INPUT# statement reads data from a file and places it into ‹ variable › names on the input list. The file must be open for reading. Data transfer for each ‹ variable › on the input list is terminated by encountering a carriage return character (CHR$(13)) or comma (CHR$(44)) or a line-feed character (CHR$(10))/or a null character (CHR$(0)).

Up to 80 characters of string or numeric data may be read into each variable. If more than 80 characters are read without encountering a carriage return, comma, or line-feed, an error condition results.

Syntax — INPUT#‹ file number ›,‹ variable ›[,‹ variable ›]...[‹,variable ›]

Example:

```
INPUT#2,A$
```

This reads data from logical file number 2 and assigns it to the variable A$. More than one variable can be read using the INPUT# statement. A comma must precede each variable name in the input list.

Example:

```
INPUT#6,A$,B$,C$
```

Each variable must have been followed by a carriage return, comma or line-feed character when written to disk to be read separately. Up to 80 characters are transferred into each input variable.

The 80 character limitation of INPUT# can be bypassed by using the GET# command which is described next.

GET# Read a character from a file

The GET# command is used to read one character of data from the disk. The file must have been opened for reading with the DOPEN# or OPEN command. The data read can be string or numeric.

Syntax — GET#‹ file number ›,‹ variable ›

Example:

```
GET#8,A
```

This reads one character from logical file number 8 into numeric variable A.

When reading string data, a binary zero (or null character) must be converted to a CHR$(0) for proper handling by BASIC.

```
GET#8,B$:IF B$= "" THEN B$= CHR$(0)
```

This puts the required binary zero in the string variable.

The GET# command allows continuous reading of the data from the file. This is done by concatenating the characters read into a new variable as follows:

```
BIG$= "": REM          initialize result
FOR I=1 TO 200: REM    start loop
GET#18,A$: REM         read one byte
BIG$= BIG$+ A$: REM    add to final result
NEXT: REM              increment loop
```

At the end of the loop 200 characters have been read into BIG$. A maximum of 255 characters can be read into a single string-type variable in this manner.

RECORD# Position the file access pointer

This command allows absolute access to any position in a file. It is used only with relative access files. If ‹ byte position › is omitted the file access pointer is set to the first character of a record.

Syntax — RECORD#‹ file number ›,‹ rec number ›[,‹ byte position ›]

Example:

```
RECORD#16, 10, 20
```

The example positions the pointer for ‹ file number › 16 at the 20th byte of the 10th record.

Example:

```
RECORD#2,25
INPUT#2,A$
```

This reads up to 80 characters from record 25 of file number 2 into the variable A$.

RECORD# is discussed in greater detail in chapter 5.

CHAPTER 5. ADVANCED FILE HANDLING

CONTENTS

CHAPTER 5. ADVANCED FILE HANDLING

RELATIVE FILES

Direct access (or Relative files) is a method that allows the programmer to position a pointer 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 and it becomes apparent that such relative handling method would result in a great reduction in the amount of time required to find and fetch a specific record stored on disk.

The three main components of a relative file are the super side sector (DOS 2.7 and 3.0 only), the side sector chain of blocks and the data block chain. All are linked together through forward pointers similar to those used in a sequential file.

The super side sector points to the first side sector in a group of side sectors. Each side sector points to the other side sectors in the same group and points to the data block chain. Record sizes, while fixed in length for each data file, may range from 1 to 254 bytes. The number of records is limited (under DOS 2.1 and DOS 2.5) to that which can be contained in 720 data blocks, as each of the six side sectors can contain a maximum of 120 data block pointers. The number of records under DOS 2.7 and DOS 3.0 is limited by the capacity of the disk but for practical purposes may not exceed 65,535 records.

The side sectors do not contain record information, but do contain pointers to the data blocks. The record size dictates where the pointer is positioned when a record number is referenced because the record size is used in an algorithm to compute where the pointer is positioned when a record number is given in the RECORD# command.

The side sector also contains a table of pointers to all of the other side sectors within a file. In order to move from one side sector to another, the pointer is referenced through the appropriate DOS command, and the corresponding side sector is read into memory. Using information contained in the referenced side sector, the data block pointer can be located and used to read the actual data block containing the record.

The relative file data block pointers in the side sectors allow 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 compared to the sequential methods.

Each side sector contains pointers for 1 to 120 data blocks. There are six side sectors for each relative file under DOS 2.1 (4040), DOS 2.5 (8050), and DOS 2.6 (2031). This provides a total file capacity of 182,880 bytes (120 pointers/side sector * six side sectors * 254 bytes per data block). The super side sector of DOS 2.7 (8250) and DOS 3.0 (hard disks) has the capacity to point to 127 groups of six side sectors, giving a potential capacity of 23,225,760 bytes per file (182,880 bytes * 127 groups of side sectors).

Spanning of data blocks 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 of DOS improves time utilization efficiency, we need to examine how I/O channels are utilized by relative files:

When a channel is opened to a previously existing file, DOS positions the file access pointer to the first record provided that the given parameters match properly. The record length variable is not necessary on the DOPEN# statement if the file already exists. DOS checks the record size (if specified) against the record size that is stored in the directory entry for an existing file. If these two do not match, an error message is generated.

Relative files require three memory buffers from the system, whereas sequential files require only two. Since there are twelve buffers in the system, and two of these are used for directory searches and internal functions, only three relative files can be open at once. 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.

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 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 span across data blocks. The only exceptions are record sizes of 1, 2, 127, and 254. These divide evenly into the 254 byte 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 via the PRINT# statement, the data block is not immediately written to disk. 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.

Because of the spanning feature, it is imperative that multiple channels NOT be open to a single relative file at the same time if any channel will be writing to the file. An update may be made in one channel's particular memory buffer area, but the change may not be made on disk until DOS moves beyond that particular data block. DOS places no restrictions on this, and when the file is open for read only, it may be advantageous to have multiple channels open to a single relative file.

DOS terminates printing to a record by detecting the EOI signal which is generated with each PRINT# statement. If the PRINT# statement goes over the maximum record size an error message is generated. Any data overflow is truncated to fit the number of characters specified by the record size and DOS position the file access pointer to the next record in the sequence.

If the PRINT# statement sends less characters than the actual record size, the remaining positions within that record are filled with nulls or binary zeroes. Consequently, when positioning to a record for input the IEEE EOI (End-Or-Identify) signal is generated from DOS to the computer when the last non-null byte is transmitted. Should the programmer desire to store binary information, a record terminator such as carriage return must be used and the record size increased by one character to accommodate the terminator.

CREATING A RELATIVE FILE

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 DOS reserves sufficient space on the disk for the future data. A relative file may be initialized by first opening the file, setting the file access pointer to the last (highest) record number to be contained in the file, printing to that record, and then closing the file.

Example:

```
DOPEN#1,d0,"FILE1",L50
RECORD#1, 100
PRINT#1, CHR$(255)
DCLOSE#1
```

In the preceding example the DOPEN# creates a file on drive 0 with the name "FILE1" and a record length of 50.

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

The PRINT# statement causes record number 100 to be written. During this write operation, DOS detects that records 1 thru 99 do not already exist, and automatically initializes them by placing a CHR$(255) in the first character of each record. During this process, all necessary side sectors and data block pointers are also created.

While DOS is generating new data blocks for relative files, the number of blocks required by the requested record number is compared to the number of data blocks left on the disk. If the resulting number of data blocks is greater than the number available on the disk, then error 52 FILE TOO LARGE is generated.

The DCLOSE# statement closes the file and causes space to be allocated in the BAM and updates the block count in the file's directory entry.

The "file setup" process described here is not a requirement since DOS will automatically create new data block pointers and side sectors as necessary. However, this procedure is recommended to increase execution speed of programs using relative files. The setup of a file in this manner need be done only once when the file is created.

EXPANDING A RELATIVE FILE

To expand an existing file, the same procedure as for creation is used, with the record number changed to reflect the greater number of records.

When DOPEN# is used on an existing relative file, the record length parameter is optional. If present, it must match the length set at the time the file was created or an error 50 RECORD NOT PRESENT results.

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

ACCESSING A RELATIVE FILE

NOTE: See Appendix E for more information regarding relative file access for writing.

In order to make the relative file handling system practical, the user must be able to access the file for reading or 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 or read from a predetermined record in a file, the RECORD# statement is used to set the DOS file access pointer to the desired record. The record number parameter may be a constant or a BASIC variable name enclosed in parenthesis as shown.

Example:

```
DOPEN#1,d0,"FILE1"
RECORD#1,25
```

or

```
RECORD#1,(rn)                Where rn has the value of 25.
PRINT#1,"Philadelphia"
DCLOSE#1
```

The resulting record would appear as follows:

(numbers added for clarity)

```
        1         2         3         4         5
12345678901234567890123456789012345678901234567890
Philadelphia*
```

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

The following program illustrates an optional feature of the RECORD# statement which permits access to individual bytes within a record for reading or writing.

```
DOPEN#1,d0,"FILE1"
RECORD#1,25      (sets file access pointer to record 25)
PRINT#1,"Philadelphia"
RECORD#1,25,20  (sets character pointer to position 20)
PRINT#1,"Penna."
RECORD#1,25,30  (sets character pointer to position 30)
PRINT#1,"19204"
DCLOSE#1
```

The following illustration is a representation of the contents of the record number 25 after the above example is executed:

(numbers added for clarity)

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
Philadelphia*      Penna.*   10204*
```

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

NOTE: It is important that the fields be written in sequence, since writing to a byte at the beginning of the record destroys the rest of the record in DOS memory. This means while it is possible to position and write first to byte 1 and then 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 in the preceding example may be retrieved by the following statements:

```
DOPEN#1,d0,"FILE1"
RECORD#1,25
INPUT#1,A$              (reads "Philadelphia" into the variable A$)
RECORD#1,25,20
INPUT#1,B$              (reads "Penna." into the variable B$)
RECORD#1,25,30
INPUT#1,C$              (reads "19204" into the variable C$)
```

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 variable D$ as an array which contains 100 elements. These elements are to be written to the disk in records number 1 thru 100 of file "FILE1". This could be accomplished with the following program.

```
DOPEN#1,d0,"FILE1"
FOR I = 1 TO 100
PRINT#1,D$(I)
NEXT I
DCLOSE#1
```

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 DOS automatically positions the file access pointer to the next record after each PRINT# statement. Therefore, the contents of the D$ array elements will be written to records 1 thru 100 of the file.

RELATIVE FILES IN 8250, D9060 and D9090 DISK UNITS

Relative files on 8050 disk units are limited to a size of 182,880 bytes. On 8250, D9060 and D9090 disk units with this limit no longer applies and relative files may use the entire capacity of a drive. The 8250 will power-up with the Expanded Relative File feature enabled. To read/write 8050 formatted relative files (on 8050 diskette) in an 8250 drive, this feature must be disabled as follows:

```
OPEN 15,8,15
PRINT#15,"M-W"CHR$(164)CHR$(67)CHR$(1)CHR$(255)
CLOSE 15
```

This disables access to expanded relative files until the 8250 drive is powered down or reset by a U: or UJ User command, or until the Expanded Relative File feature is re-enabled as follows:

```
OPEN 15,8,15
PRINT#15,"M-W"CHR$(164)CHR$(67)CHR$(1)CHR$(0)
CLOSE 15
```

Existing relative files in any diskette format can be converted to the 8250 or hard disk Expanded Relative File format by means of a program named "EXPAND.REL" which is included on the TEST/DEMO diskette supplied with the 8250 disk units. To convert other relative files to Expanded format DLOAD and RUN this program (you must have disk units of both types attached to the computer). A series of instructions is displayed on the screen. The expanded relative files written by this program can only be accessed by an 8250, D9060 or D9090 disk unit.

USING 8050 DISKETTES IN 8250 DRIVES

Although the 8050 and 8250 disk units are read/write compatible, the first access to an 8050 diskette inserted into an 8250 drive (or use of an INITIALIZE command) causes an error 66 ILLEGAL TRACK OR SECTOR message. The message occurs because of the different BAM contents of the two disk systems and may be ignored. The error only occurs once and all further disk commands operate correctly unless the diskette is moved to another drive.

For ease of use, data on 8050 diskettes should be transferred to 8250 formatted diskettes using the COPY command. The BACKUP command cannot be used.

The 8050 disk unit is upward compatible (read/write) to the 8250 with some exceptions. The 8050 disk unit cannot access the reverse (top) side of an 8250 formatted diskette. Relative files created on an 8250 disk unit cannot be accessed by an 8050 unless the Expanded Relative File feature of the 8250 was disabled before creating the file and unless the file resides entirely on the bottom diskette surface, the one that the 8050 accesses.

CHAPTER 6. ADVANCED DOS PROGRAMMING

CONTENTS

CHAPTER 6. ADVANCED DOS PROGRAMMING

DOS OVERVIEW DESCRIPTION

DOS 2.1 works with the 4040 dual disk unit. Model 2040 disk units (with DOS 1.0) can be upgraded to DOS 2.1 by replacement of ROM chips within the disk unit. Reliability of the recording format of DOS 2.1 was improved over DOS 1.0 by removing one block from tracks 18 thru 24. As a result the directory holds 144 file entries and 664 blocks for user data.

The Relative Record file structure was added to DOS 2.1 to provide for random access to files. The Block Read/Write commands of DOS 1.0 are supported, but the corresponding "U1" and "U2" utility commands should be used for upward compatibility with future CBM disk products.

In general, software which does not depend on physical device attributes should be upward compatible with all versions of DOS. Programs using the Block Read/Write commands are very vulnerable to DOS changes.

DOS 2.5 is used in all 8050 dual disk drive units. All of the features of DOS 2.1 are included in DOS 2.5 and adapted for additional capacity. DOS 2.5 also includes enhancements such as disk insertion detect and expanded error recovery techniques. The directory provides 224 file entries and 2052 blocks are available for user data.

DOS 2.6 is used with the 2031 single disk unit. DOS 2.6 is a functional equivalent to DOS 2.1 (used in the 4040) and is fully compatible with DOS 2.1 with one exception. Since the 2031 is a single drive unit, dual drive commands will not work on the 2031, and will cause a drive not ready error.

DOS 2.7 is used in the 8250 double-sided dual disk unit. DOS 2.5 disk commands and the 8050 disk unit are upward compatible with DOS 2.7 and the 8250. With certain restrictions diskettes created on either disk unit are read/write compatible. One important feature of the 8250 is the Expanded Relative File capability of DOS 2.7 which allows relative files to occupy an entire 8250 diskette, providing over 1 million bytes of storage.

DOS 3.0 is used in the D9060 and D9090 hard disk units. Features of DOS 3.0 include a dynamically expandable directory allowing an unlimited number of file entries, replacement-mapping of bad sectors, and a self-locating BAM. Relative files may occupy the entire capacity of the hard disk unit.

General Operation of DOS

The DOS file interface controller is responsible for managing all data transfers between the IEEE bus and the disk controller. Most disk I/O is performed on a pipeline basis, resulting in faster response to requested operations.

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

The common memory between the disk controller and the file interface is used for 256-byte buffer areas. Three of the sixteen buffers are used by DOS for the Block Availability Map (BAM), variable space, command channel I/O, and the disk controller job queue.

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. A status byte 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 on 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 DLOAD and DSAVE statements transmit secondary addresses of 0 and 1, respectively. DOS automatically interprets these secondary addresses as DLOAD and DSAVE functions. Unless the 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.

DISK UTILITY COMMAND SET

The disk utility command set consists of the following commands. The format and operation of these commands is compatible across all versions of DOS.

COMMAND	ABBREVIATION	GENERAL FORMAT
BLOCK—ALLOCATE	B—A	"B—A:"dr;t;s
BLOCK—FREE	B—F	"B—F:"dr;t;s
BLOCK—READ	B—R	"B—R:"ch;dr;t;s
BLOCK—WRITE	B—W	"B—W:"ch;dr;t;s
BLOCK—EXECUTE	B—E	"B—E:"ch;dr;t;s
BUFFER—POINTER	B—P	"B—P:"ch;p
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	"Ux:"ch;dr;t;s

The format conventions for the DOS utility commands is shown below:

ch = the channel number in DOS, identical to the secondary address in the associated OPEN statement.

dr = the drive number, 0 (or 1 for dual drive units).

t = the track number, 1 thru 154 (depending on model).

s = the sector number, 0 thru 112 (depending on model).

p = the pointer position for the buffer pointer.

adl = the low byte of the address in chr$(n) form.

adh = the high byte of the address in chr$(n) form.

nc = the number of characters, 1 thru 34, in chr$(n) form.

data = the actual data in hexadecimal. This is trasmitted by using the chr$ function, i.e. chr$(17) would send the hexadecimal equivalent of decimal 17.

x = the index into the User Table.

These commands may be abbreviated to the first character of each of the key words. Only abbreviations are accepted for the Memory Read, Write, and Execute commands. DOS searches for parameters associated with each command starting at a colon (:), or in the fourth character position if a colon is not present. The following example shows four ways that the same BLOCK—READ command may be given.

Examples:

```
"BLOCK—READ:"2,1,4,0
"B—R"2,1,4,0
"B—R"2;1;4;0
"B—READ"A;B;C;D
```

Parameters following the key words within quotation marks may be separated by any combination of ‹ cursor right ›, SPACE, or comma characters. If using variable names to pass command parameters, only the command string should be enclosed in quotes as shown in the general format examples above.

The Disk Utility commands are sent to the disk unit by first opening a logical file to the DOS "command channel" using secondary address 15. Then the PRINT# statement is used to issue one or more commands. Finally, when all processing is done, the CLOSE statement is used to 'shut-down' the command channel. The example below shows the actual statements to use. In examples on the following pages, the OPEN, PRINT# and CLOSE statements are omitted for clarity.

Example:

```
OPEN 15,8,15
PRINT#15, "B-R:" 2;1;18;0
PRINT#15, "B-A:" 1;1;1
CLOSE 15
```

DISK ORIENTED UTILITIES

BLOCK—ALLOCATE

This command causes DOS to flag the block specified on the drive, track and sector to be "in use". If successful, the appropriate Block Availability Map (BAM) is updated in DOS memory to reflect the block as allocated (used). In future operations, DOS skips over the allocated block when saving programs or writing files. The updated BAM is written to disk upon closing an output file or closing the command channel.

If the block requested has been previously allocated, the error channel indicates the next available block (increasing track and sector numbers) with the NO BLOCK error. If no blocks are available, greater in number than the one which was requested, zeroes are shown as the track and sector parameters when the NO BLOCK error is returned.

Format:

"B—A:"dr;t;s

Example:

"B—A:"0;10;5

This requests that block (sector) 5 of track 10 on drive 0 be flagged as allocated on the disk. Always check the error channel when using this command to prevent an allocated block from being overwritten. If the block is already allocated, the error message also indicates the next available block.

Example:

```
OPEN 15,8,15
INPUT#15,EN,EM$,ET,ES
CLOSE 15
```

This reads the next available track and sector, respectively, into ET and ES.

EN = Error Number
EM$ = Error Message
ET = Error Track
ES = Error Sector

BLOCK—FREE

This command causes DOS to return the specified block to the pool of available storage. The block is marked as "free" in the appropriate BAM in DOS memory and may later be allocated for other uses. The BAM is written onto the disk when the command channel or any other file is closed:

Format:

"B—F:"dr;t;s

Example:

"B—F:"1;22;9

This causes DOS to free sector 9 of track 22 on drive 1.

Care must be taken when using this command to avoid track and sector values belonging to the BAM, Directory Header, or Directory. If any of these blocks are freed, DOS may allocate them for other use, with catastrophic results.

BLOCK—READ

This disk utility command provides direct access to any block on the disk. Used in conjunction with other block commands, a random access file system may be created through BASIC. This command positions the DOS file access pointer to the first character or "0-position" of the block. When a character in this position is read with the 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 next GET# or INPUT# will read the data normally.

Format:

"B—R:"ch;dr;t;s

Example:

"B—R:"5;0;18;0

This reads the block from drive 0, 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., by using that secondary address). The U1 command described under User is similar to the BLOCK—READ command.

NOTE: The B-R command can cause errors in 4040/2031 operations. For 4040 and 2031 models, the corresponding "U1:" command (see page 59) should *always* be used instead.

BLOCK—WRITE

When this command is initiated, the current buffer pointer is used as the last character pointer and is placed in the sector-link position of the buffer. The buffer is then written to the indicated block on the disk and the buffer pointer is set to position 1.

Format:

"B—W:"ch;dr;t;s

Example:

"B—W:"7;0;35;10

This writes channel 7 buffer to the block on drive 0, track 35, sector 10. The U2 command described under User is similar to the BLOCK—WRITE command.

NOTE: The B-W command can cause errors in 4040/2031 operations. For 4040 and 2031 models, the corresponding "U2:" command (see page 59) should *always* be used instead.

BLOCK—EXECUTE

This command allows part of the DOS or user designed routines to reside on disk, be loaded into disk drive memory, and be executed. The File Interface Controller begins execution of the contents after the block is read into the specified buffer. Execution must be terminated with a "Return From Subroutine" (RTS) instruction. Future system extensions or user-created functions may implement this feature.

Format:

"B—E:"ch;dr;t;s

Example:

"B—E:"6;0;1;10

This reads a block from drive 0, 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 a given channel buffer to a new value. This is useful when accessing particular fields of a record within a block, or if the block is divided into records, individual records may be set for transmitting or receiving data.

Format:

"B—P:"ch;p

Example:

"B—P:"2;10

This sets the channel 2 buffer pointer to the byte position 10 of the data area in the DOS memory buffer. The buffer pointer value may range from 0 thru 255.

MEMORY UTILITIES

All three Memory commands are byte-oriented so that the user may utilize machine language programs. BASIC statements may be used to access data via Memory commands by using the CHR$ function. The system accepts only M—R, M—W, and M—E; neither the full spelling or the use of a colon (:) is permitted. The INITIALIZE command must be sent (only once) to a drive before issuing a sequence of Memory commands to the drive.

MEMORY—WRITE

This command provides direct access to DOS memory. Special routines may be down-loaded to the disk drive via this command and then executed by using the MEMORY—EXECUTE command or one of the User commands. Up to 34 bytes may be deposited with each use of the M—W command. The hexadecimal value of DOS memory address must be specified low-byte first and must be converted to decimal for use with the CHR$ function.

Format:

"M—W'adl adh nc data (spaces are shown for clarity)

Example:

"M—W"CHR$(0)CHR$(18)CHR$(4)CHR$(32)CHR$(0)CHR$(17)CHR$(96)

This write four bytes to buffer 2 ($1200 or decimal 4608). Another use for the M—W command is to temporarily change the physical device number of a disk unit. All disk units are set to device number 8 at the factory.

When two or more disk units are attached to the computer, the device number of each unit must be made unique or none will operate correctly. The following program fragment changes the device number of 4040, 8050, or 8250 floppy disk units and the D9060 or D9090 hard disk units.

Example: ("odn" is old device number, "ndn" is new device number)

```
        OPEN15,odn,15
PRINT#15, "M—W"CHR$(12)CHR$(0)CHR$(2)CHR$(ndn + 32)CHR$(ndn + 64)
        CLOSE15
```

To change the device number of 2031 disk units use this M—W statement:

Example:

```
PRINT#15,"M—W"CHR$(119)CHR$(0)CHR$(2)CHR$(ndn + 32)CHR$(ndn + 64)
```

The general procedure to change device number is:

1. power-up the first disk unit only.
2. run the above program.
3. if the next unit is a 2031 disconnect it from the IEEE bus.
4. power-up the next disk unit.
5. if this unit is a 2031 re-connect it to the IEEE bus and proceed to step 2.

The device number remains at the new value until changed again by the M—W command or a (U: or UJ) command is issued or the unit is powered down.

MEMORY—READ

The single byte pointed to by the DOS memory address in the command string may be accessed with this command. Also, variables from DOS or the contents of the buffers may be read with this command. The M—R command changes the contents of the error channel since that is used for transmitting data to the computer. The next GET# from the error channel (secondary address 15) transmits the byte.

An INPUT# should not be executed after a MEMORY—READ command until after the error channel has been closed or until a DOS command other than one of the Memory commands is executed. This is because the unchanged data byte in the error channel would be interpreted as an error number.

Format:

"M—R" adh adl (spaces shown for clarity)

Example:

"M—R"CHR$(128)CHR$(0)

This accesses and reads the byte located at $0080 hexadecimal or 128 decimal. The following GET# transfers the data to the computer.

MEMORY—EXECUTE

Subroutines in DOS memory may be executed with this command. To return to the DOS, terminate the subroutine with a RTS instruction.

Format:

"M—E" adl adh (spaces shown for clarity)

Example:

"M—E"CHR$(128)CHR$(49)

This causes execution of the code beginning at location $3180 hexadecimal. The most common use of the "M—E" command is to first create a subroutine in a specific buffer and then call it for execution as needed with the "M—E" command. The subroutine may be "downloaded" from the computer using the print #15, "M—W" statements or may be read into a buffer using the U1 command. Great caution should be taken in using "M—E" because an incorrect execution address could cause destruction of your data.

USER UTILITIES

These commands provide a link to 6502 machine code in the disk according to a jump table pointed to by the special User pointer. The second character in this command is used as an index to the table. The ASCII characters 0 thru 9 or letters A thru J 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 on all DOS versions. Because of errors in DOS 2.1 the B—R and B—W commands do not operate correctly in 4040 disk units. Thus B—R and B—W must be replaced with U1 and U2 when programming the 4040.

The U1 command 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, including the track and sector link pointer.

Format:

"U1:"ch;dr;t;s

Example:

"U1:"5;0;18;0

This causes the block at track 18, sector 0, drive 0, to be read into the buffer channel number 5. The data may then be accessed using the M—R and GET# commands.

U2 writes a buffer to a block on the disk without changing the data block link pointer 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 disk with U2.

Format:

"U2:"ch;dr;t;s

Example:

"U2:"5;0;18;0

This writes the data in channel buffer number 5 to drive 0, track 18, sector 0.

STANDARD USER JUMP TABLE

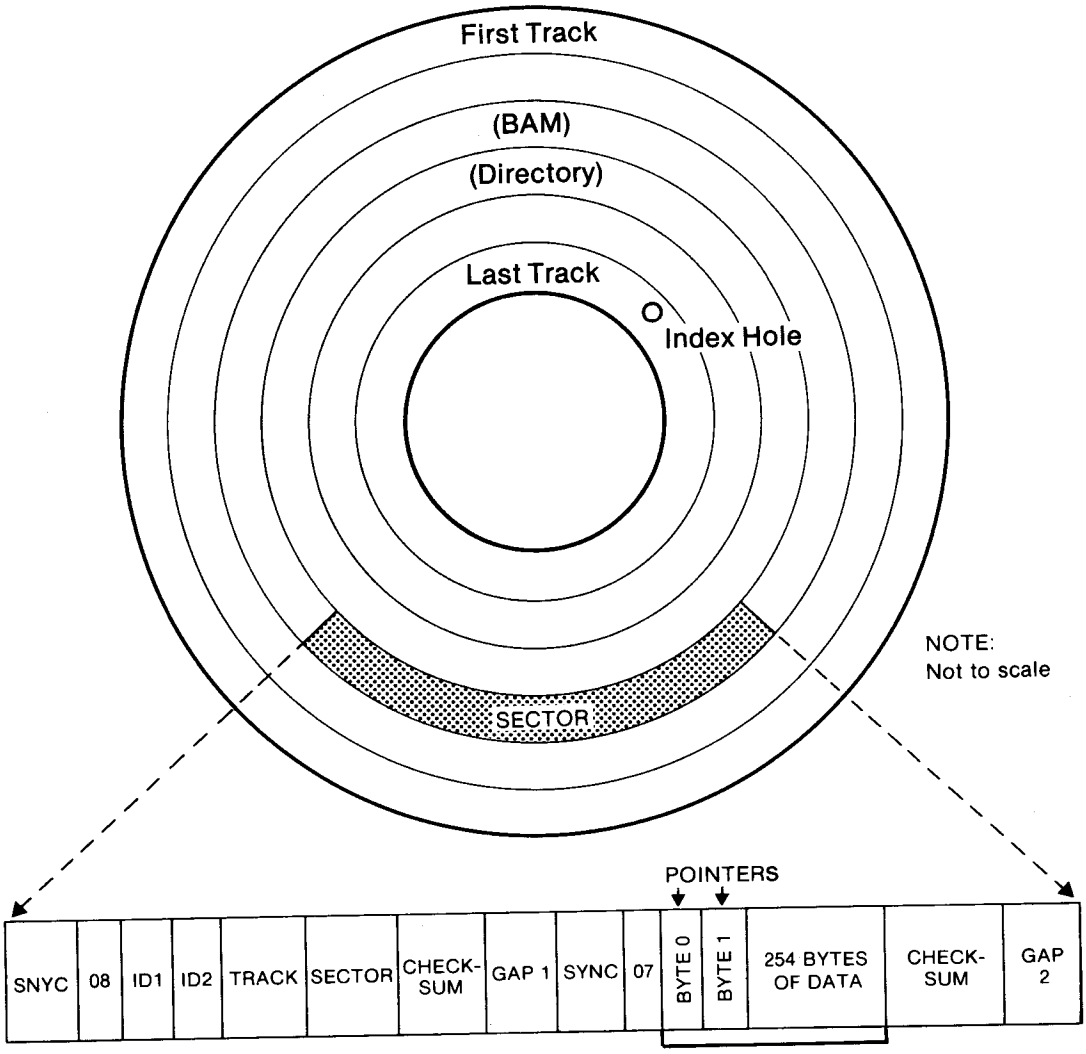
STANDARD DESIGNATION	ALTERNATE DESIGNATION	FUNCTION
U1	UA	BLOCK—READ replacement
U2	UB	BLOCK—WRITE replacement
U3	UC	jump to $1300
U4	UD	jump to $1303
U5	UE	jump to $1306
U6	UF	jump to $1309
U7	UG	jump to $130C
U8	UH	jump to $130F
U9	UI	jump to $10F0 (NMI)
U:	UJ	Power-up Vector

The U3 thru U8 commands are provisions for user-defined commands. The locations jumped to would contain jump instructions to subroutines located in the buffer areas of disk unit RAM. User-written DOS routines may be coded to reside there and may be down-loaded from the computer using the M—W command or read from disk using the B—R or U1 commands. The new DOS command can then be called by using the appropriate U3 thru U8 command. The U: or UJ commands cause the disk drive to perform its power-up sequence and resets the device number to 8. The drive(s) must be initialized before using further commands.

CHAPTER 7. DISK STORAGE FORMATS

This chapter provides the details of disk storage formats of the 2031, 4040, 8050, and 8250 floppy disk units and the D9090 and D9060 hard disk units. For each type of the disk the tables which follow show: Block distribution by track, locations and formats of the Block Allocation Map, the Directory Header, the Directory, and the formats of Program, Sequential, and Relative files.

CONTENTS



Typical Disk Format
FIGURE 7.1

Figure 7.1 is a graphic representation of a disk format. The tracks are shown as concentric rings, with track 1 being on the outside of the disk. The number of tracks on a disk varies with the model of the disk unit. The number of sectors on each track varies with model and track number as shown in the following tables.

A sector has information that enables the DOS hardware to access and verify each sector for integrity. These items do not concern a programmer, and are normally not accessible. The items of importance are the pointers and the data.

Some sectors are used for various purposes by DOS, such as Directory Header, Directory, and BAM information. For each disk unit the locations and formats of these sectors are described. Data file formats (common to all models) are also described in this chapter.

2031 DISK UNIT

NUMBER OF BLOCKS PER TRACK

TRACK NUMBER	NUMBER OF BLOCKS
1 to 17	21
18 to 24	19
25 to 30	18
31 to 35	17

BLOCK ALLOCATION MAP FORMAT Track 18 Sector 00

BYTE	DATA	DESCRIPTION
0-1	18-00	Track-Sector of first directory block
2	65	ASCII "a" identifies DOS 2.6 format
3	00	reserved for future use
4-143		Bit map of available blocks, tracks 1-35

DIRECTORY HEADER FORMAT Track 18 Sector 00

BYTE	DATA	DESCRIPTION
0-143		reserved for BAM
144-161		Diskette name, padded with shifted spaces
162-163		Diskette ID number
164	160	Shifted space
165-166	50,65	ASCII "2a" identifies DOS version & format
167-170	160	Shifted spaces
171-255	00	not used

4040 DISK UNIT

NUMBER OF BLOCKS PER TRACK

TRACK NUMBER	NUMBER OF BLOCKS
1 to 17	21
18 to 24	19
25 to 30	18
31 to 35	17

BLOCK ALLOCATION MAP FORMAT Track 18 Sector 00

BYTE	DATA	DESCRIPTION
0-1	18-00	Track-Sector of first directory block
2	65	ASCII "a" identifies DOS 2.1 format
3	00	reserved for future use
4-143		Bit map of available blocks, tracks 1-35

DIRECTORY HEADER FORMAT Track 18 Sector 00

BYTE	DATA	DESCRIPTION
0-143		reserved for BAM
144-161		Diskette name, padded with shifted spaces
162-163		Diskette ID number
164	160	Shifted space
165-166	50,65	ASCII "2a" identifies DOS version & format
167-170	160	Shifted spaces
171-255	00	not used

Note: ASCII data may appear in bytes 180-191 on some diskettes.

8050 DISK UNIT

NUMBER OF BLOCKS PER TRACK

TRACK NUMBER	NUMBER OF BLOCKS
1 to 39	29
40 to 53	27
54 to 64	25
65 to 77	23

BLOCK ALLOCATION MAP (First Block) FORMAT Track 38 Sector 00

BYTE	DATA	DESCRIPTION
0-1	38-03	Track—sector of second BAM block
2	67	ASCII "c" identifies DOS 2.5 format
3	00	reserved for future use
4	01	Lowest track # mapped in the BAM block
5	51	Highest track # (+ 1) mapped in this BAM block
6-255		Bit map of available blocks on tracks #1-50

BLOCK ALLOCATION MAP (Second Block) FORMAT Track 38 Sector 03

BYTE	DATA	DESCRIPTION
0-1	39-01	Track-Sector of first directory block
2	67	ASCII "c" identifies DOS 2.5 format
3	00	reserved for future use
4	51	Lowest track # mapped in 2nd BAM block
5	51	Highest track # (+ 1) mapped in this BAM block
6-140		Bit map of available blocks on tracks #51-77
141-255	00	not used

DIRECTORY HEADER FORMAT Track 39 Sector 00

BYTE	DATA	DESCRIPTION
0-1	38-00	Track-Sector pointer to first BAM block
2	67	ASCII "c" identifies DOS 2.5 format
3	00	reserved for future use
4-5		not used
6-21		Diskette name, padded with shifted spaces
22-23	160	Shifted spaces
24-25		Diskette ID number
26	160	Shifted space
27-28	50,67	ASCII "2c" identifies DOS version & format
29-32	160	Shifted spaces
33-255	00	not used

8250 DISK UNIT

NUMBER OF BLOCKS PER TRACK

TRACK NUMBER	NUMBER OF BLOCKS
1 to 39	29
40 to 53	27
54 to 64	25
65 to 77	23
78 to 116	29
117 to 130	27
131 to 140	25
142 to 154	23

BLOCK ALLOCATION MAP (First Block) FORMAT Track 38 Sector 00

BYTE	DATA	DESCRIPTION
0-1	38-03	Track-Sector of second BAM block
2	67	ASCII "c" identifies DOS 2.7 format
3	00	reserved for future use
4	01	Lowest track # mapped in the BAM block
5	51	Highest track # (+ 1) mapped in this BAM block
6-255		Bit map of available blocks on tracks #1-50

BLOCK ALLOCATION MAP (Second Block) FORMAT Track 38 Sector 03

BYTE	DATA	DESCRIPTION
0-1	39-06	Track-Sector of third BAM block
2	67	ASCII "c" identifies DOS 2.7 format
3	00	reserved for future use
4	51	Lowest track # mapped in 2nd BAM block
5	101	Highest track # (+ 1) mapped in 2nd BAM block
6-255		Bit map of available blocks on tracks #51-100

BLOCK ALLOCATION MAP (Third Block) FORMAT Track 38 Sector 06

BYTE	DATA	DESCRIPTION
0-1	38-09	Track-Sector of fourth BAM block
2	67	ASCII "c" identifies DOS 2.7 format
3	00	reserved for future use
4	101	Lowest track # mapped in the BAM block
5	151	Highest track # (+ 1) mapped in this BAM block
6-255		Bit map of available blocks on tracks #101-150

BLOCK ALLOCATION MAP (Fourth Block) FORMAT Track 38 Sector 09

BYTE	DATA	DESCRIPTION
0-1	39-01	Track-Sector of first directory block
2	67	ASCII "c" identifies DOS 2.7 format
3	00	reserved for future use
4	151	Lowest track # mapped in 4th BAM block
5	155	Highest track # (+ 1) mapped in 4th BAM block
6-25		Bit map of available blocks on tracks #151-154
26-255	00	Not Used

DIRECTORY HEADER FORMAT Track 39 Sector 00

BYTE	DATA	DESCRIPTION
0-1	38-00	Track-Sector pointer to first BAM block
2	67	ASCII "c" identifies DOS 2.7 format
3	00	reserved for future use
4-5		not used
6-21		Diskette name, padded with shifted spaces
22-23	160	Shifted spaces
24-25		Diskette ID number
26	160	Shifted space
27-28	50,67	ASCII "2c" identifies DOS version & format
29-32	160	Shifted spaces
33-255	00	not used

D9060/D9090 DISK UNIT

D9060	4 Recording Surfaces
D9090	6 Recording Surfaces

153 Tracks per Recording Surface
32 Sectors per Track

BLOCK ALLOCATION MAP FORMAT Track 1 Sector 0 (normally)

BYTE	DATA	DESCRIPTION
0-1		Track-Sector pointer to next BAM block ($FFFF = last BAM block)
2-3		Track-Sector to previous BAM block ($FFFF = first BAM block)
4		Lowest track # mapped in this BAM block
5		Highest track # (+ 1) mapped in this BAM block
6-255		Bit map of up to 50 tracks, for one recording surface

DIRECTORY HEADER FORMAT Track 0 Sector 0

BYTE	DATA	DESCRIPTION
0-1	* 00-01	Track-Sector pointer to Bad Track & Sector List
2-3	00-255	Identifies DOS 3.0 format
4-5	* 76-00	Track-Sector of first directory block
6-7	* 76-20	Tract-Sector of Header Sector
8-9	* 01-00	Track-Sector of first BAM block
10-11	ID-ID	Disk ID Code
12-255	00	not used

* The values for track and sector locations will vary due to dynamic reallocation.

FORMATS COMMON TO ALL DISKS

BAM BLOCK FORMAT

Each track has five bytes allocated to map it. A map bit of 1 means that the block is available. A map bit of 0 means the block is presently allocated. Blocks are mapped by bytes, the high order bit of each byte maps the lowest numbered block of each group.

BYTE	Definition		
1	Current number of available blocks for this track		
2	Bit map of blocks 0 - 7.	Bit 7 = block 0,	bit 0 = block 7
3	Bit map of blocks 8 - 15.	Bit 7 = block 8,	bit 0 = block 15
4	Bit map of blocks 16 - 23.	Bit 7 = block 16,	bit 0 = block 23
5	Bit map of blocks 24 - 31.	Bit 7 = block 24,	bit 0 = block 31

DIRECTORY BLOCK FORMAT

2031	Track 18 Sectors 01 through 18
4040	Track 18 Sectors 01 through 18
8050	Track 39 Sectors 01 through 29
8250	Track 39 Sectors 01 through 29
D9060/D9090	Starts on Cylinder 76, uses all tracks - Sectors 00 through 31, then expands to additional blocks as needed, providing 'unlimited' directory size.

BYTE	DATA	DESCRIPTION
0-1		Track-Sector pointer to next directory block
2		File type
3-4		Track-Sector pointer to first file block
5-20		File name, padded with shifted spaces
21-22		Track-Sector of first side sector (or super side sector) if Relative File
23		Record length if Relative file
24-27		reserved for future use
28-29		Track Sector pointer for replacement
30-31		Number of blocks used by the file
32-255		Seven more 32-byte file entries, same as 0-31 above, except the first two bytes of each entry are unused

NOTE:

1. 32 bytes per file entry.
2. Eight file entries per directory block.
3. File type data is:

Scratched files	$00
Sequential data	$01
Program files	$02
User-defined	$03
Relative Record	$04

4. File type data is OR'ed with $80 when file is closed.
5. Track value of 00 in byte zero indicates the last used block in the directory. Sector value then shows the next byte to use.

DISK DATA FILE FORMATS

PROGRAM FILES

BYTE	Definition
0-1	Track-Sector pointer to next program block
2-255	Up to 254 bytes of the program. End-Of-File is indicated by three consecutive bytes of $00.

SEQUENTIAL DATA RECORDS

BYTE	Definition
0-1	Track-Sector pointer to next sequential data block. A track number of 00 indicates the last data block. The sector pointer then indicates the next byte position available.
2-255	Up to 254 data bytes with a carriage return character (CHR$(13)) as delimiter between data items.

RELATIVE DATA RECORDS

BYTE	Definition
0-1	Track-Sector pointer to next data block assigned for the file. A track pointer of $00 indicates the last data block, and the sector pointer then points to the next byte position available for use.
2-255	Up to 254 data bytes with a carriage return character (CHR$(13)) as delimiter between data items.

RELATIVE FILE SIDE SECTOR FORMAT

BYTE	Definition
0-1	Track-Sector pointer to next side sector.
2	Side sector number - if 2031, 4040 or 8050 relative file - always $FE if 8250, D9060 or D9090
3	Relative record length
4-5	Track-Sector pointer - first side sector
6-7	Track-Sector pointer - second side sector
8-9	Track-Sector pointer - third side sector
10-11	Track-Sector pointer - fourth side sector
12-13	Track-Sector pointer - fifth side sector
14-15	Track-Sector pointer - sixth side sector
16-255	Track-Sector pointers to 120 data blocks
	Total of 720 blocks (maximum of 182.8K bytes) per file

DOS 2.7 and DOS 3.0 Super Side Sector contains Track-Sector pointers to 127 groups of 6 side sectors as above for a potential file size of 23.25 Megabytes.

CHAPTER 8. DOS ERROR MESSAGES

CONTENTS

CHAPTER 8. DOS ERROR MESSAGES

REQUESTING ERROR MESSAGES

The method for obtaining an error message depends upon the version of BASIC in use. After display of the error message the device error indicator is cleared.

FOR BASIC 3.0 (except 8250)

```
OPEN 1,8,15
INPUT#1,A,B$,C,D
PRINT A,B$,C,D
```

FOR BASIC 3.0 (8250 Drives DOS 2.7)

```
OPEN 1,8,15
INPUT#1,A,B$,C,D,E
PRINT A,B$,C,D,E
```

FOR BASIC 4.0

```
PRINT DS$
```

The above variables represent:

A	Message Number
B$	Error Message
C	Track
D	Sector
E	Drive Number for DOS 2.7

BASIC 4.0 also prints the above five items.

The following is a description of the error messages.

Errors 2 to 19 are unused and should be ignored.

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 (drive not ready) Indicates 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 (bad sector flag)
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.

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.

30: SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel. Typically caused by an illegal number of file names, or pattern matching illegally used.

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 illegally used in the DOPEN or DSAVE 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 via 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 to a valid record number.

51: OVERFLOW IN RECORD
Data written with a PRINT# statement exceeds the defined relative record size. Data is truncated to the defined size. Typical cause is failing to include carriage returns sent as field or record terminators in calculating the record size.

52: FILE TOO LARGE
Record position within a realtive file indicates that not enough blocks remain available on the disk to contain the specified number of records.

60: WRITE FILE OPEN
This message is generated when 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 disk.

64: FILE TYPE MISMATCH
The file type on a DOPEN command 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 be allocated has been previously allocated. The parameters indicate the next higher track and sector number available. If the parameters are zeroes then all higher numbered blocks 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
Special error message indicating 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 or three relative files may be opened at one time to the DOS. Direct access channels may have six opened files.

71: DIRECTORY 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 disk to restore the BAM in memory. Active files may be terminated by the corrective action.

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

74: DRIVE NOT READY
An attempt has been made to access an invalid device number or the disk is not powered-up or not up to speed.

75: FORMAT SPEED ERROR
While formatting diskettes the 8250 verifies that drive speed is within 2 milliseconds (1%) of being 200 milliseconds per revolution. If speed is outside that limit the formatting is halted with the disk error light on.

76: Controller error — a variety of conditions indicating controller hardware problems.

APPENDIX A. SUMMARY OF DOS COMMANDS

The quick reference guide on the following pages will assist you in becoming familiar with DOS commands as used in both BASIC 3.0 and BASIC 4.0. The commands of BASIC 3.0 are an upward compatible subset of BASIC 4.0 commands and are recognized by BASIC 4.0 with the same results.

Commands for BASIC 4.0 are shown followed by their BASIC 3.0 equivalent. For some commands only one syntax is given for all versions of BASIC. Command names are shown in lower-case letters to illustrate command abbreviations. The parameters for the commands are shown under the command name in the table.

A separate list is shown of commands available when using the "Universal Wedge" program. The Universal Wedge is a program supplied on the TEST/DEMO diskette shipped with floppy disk units. The program is loaded into computer memory and, when run, provides a set of DOS commands identical in format regardless of BASIC version. To enable these commands, load the "Universal Wedge" file from disk and type the BASIC statement: SYS 7*4096.

The "Wedge" commands may only be entered directly from the keyboard and may not be used in programs. Use of the Universal Wedge program is never required, but is provided as a convenient utility.

CBM DOS Command Quick Reference

COMMAND	ABBREVIATION	BASIC VERSION
append#	aP#	4.0
‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"		
backup	bA	4.0
d‹ drive ›[onu‹ unit ›] to d‹ drive ›[onu‹ unit ›]		
print#15,	pR15	3.0
"d[uplicate]‹ dest drive › = ‹ src drive ›"		
catalog	cA	4.0
directory	diR	4.0
[d‹ drive ›][onu‹ unit ›]		
load	l0	3.0
"$‹ drive ›",‹ unit › SEE NOTE 1.		
collect	coL	4.0
[d‹ drive ›][onu‹ unit ›]		
print#15,	pR	3.0
"v[alidate][‹ drive ›]"		
concat	conC	4.0
[d‹ drive ›][onu‹ unit ›],"‹ file name ›" to [d‹ drive ›][onu‹ unit ›],"‹ file name ›"		

copy	coP	4.0
[d‹ drive ›][onu‹ unit ›],"‹ src file ›" to [d‹ drive ›][onu‹ unit ›],"‹ dest drive ›"		
print#15,	pR	3.0
"c[opy]‹ dest drive ›:‹ dest file › = ‹ src drive ›:‹ src file ›"		
dclose#	dC#	4.0
[‹ file number ›]		
close	cl0	3.0
[‹ file number ›]		
directory	diR	4.0
(see catalog command)		
dload	dL	4.0
[d‹ drive ›][onu‹ unit ›,]"‹ file name ›"		
load	l0	3.0
["‹ drive ›:‹ file name ›",‹ unit ›]		

dopen#	dO#	4.0	Sequential Files
‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"[,R] ‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"[,W]			
open	oP	3.0	Sequential Files
‹ file number ›,‹ unit ›,‹ address ›,"‹ drive ›:‹ file name ›"[,R] ‹ file number ›,‹ unit ›,‹ address ›,"‹ drive ›:‹ file name ›"[,W]			
dopen#	dO#	4.0	Relative Files
‹ file number ›[,d‹ drive ›][onu‹ unit ›],"‹ file name ›"[,L‹ rec len ›]			
open	oP	3.0	Relative Files
‹ file number ›,‹ unit ›,‹ address ›,"‹ drive ›:‹ file name ›,L"+ chr$(‹ rec len ›)			
dsave	dS	4.0	
[d‹ drive ›][onu‹ unit ›,]"‹[@] file name ›"		See Note 2.	
save	sA	3.0	
"[@] (‹ drive ›] :‹ file name ›",‹ unit ›		See Note 2.	
header	hE	4.0	
"‹ name ›"[,d‹ drive ›][onu‹ unit ›][,i‹ id ›]			
print#15,	pR	3.0	
"n[ew][‹ drive ›]:‹ name ›[,‹ id ›]"			
initialize	I	ALL	
print#15,"i[nitialize][‹ drive ›]"			

input#	iN#	ALL
‹ file number ›,‹ variable ›[,‹ variable ›]....[,‹ variable ›]		
record#	reC#	4.0
‹ file number ›,‹ rec number ›[,‹ byte position ›]		
print#15,	pR	3.0
"P"chr$(‹ file number + 96 ›)chr$(‹ rec num low-byte ›)chr$(‹ rec num high-byte ›)[chr$(‹ byte position ›)]		
rename	reN	4.0
[d‹ drive ›][onu‹ unit ›,]"‹ original name ›" to "‹ new name ›"		
print#15,	pR	3.0
"r[ename][‹ drive ›]:‹ new name ›=‹ original name ›"		
scratch	sC	4.0
[d‹ drive ›][onu‹ unit ›,]"‹ file name ›"		
print#15,	pR	3.0
"s[cratch]‹ drive ›:‹ file name ›[,‹ drive ›:‹ file name ›]...."		

NOTE 1: This command in 3.0 BASIC destroys previous computer memory contents as the directory is loaded. To display the directory type LIST.

NOTE 2: The 'at sign' (@) creates the program file-name if it does not already exist. If a file of the same name is already on the disk, it is replaced with the program in memory.

UNIVERSAL WEDGE COMMANDS

WEDGE SYNTAX	BASIC 4.0 EQUIVALENT
SAVE"‹ drive ›:‹ file name ›",‹ unit ›	DSAVE
/‹ drive ›:‹ file name ›	DLOAD
↑‹ drive ›:‹ file name ›	Shifted RUN/STOP key
›$‹ drive ›	DIRECTORY
›(carriage return)	Request DOS Error Message
›N‹ drive ›:‹ disk name ›,id	HEADER
›I‹ drive ›	INITIALIZE
›V‹ drive ›	COLLECT
›C‹ dest drive › = ‹ src drive ›	COPY (all files)
›C‹ dest drive ›:‹ dest file › = ‹ src drive ›:‹ src name ›	COPY (one file)
›R‹ drive ›:‹ new name › = ‹ original name ›	RENAME
›S‹ drive ›:‹ file name ›	SCRATCH

SUMMARY OF COMMANDS AND STATEMENTS

APPEND# —

Open a sequential file to continue output

BACKUP —

Duplicates the entire contents of a disk

BLOCK-ALLOCATE —

Marks the requested block as "in use"

BLOCK-FREE —

Marks the requested block as "free" for use

BLOCK-EXECUTE—

Begins execution of 6502 machine code within the disk unit

BLOCK-READ —

Provides direct access to any block on the disk

BLOCK-WRITE —

Writes a DOS memory buffer to the requested block on the disk

BUFFER-POINTER —

Changes character pointer to a new value

CATALOG —

Displays a list of file names on the disk

COLLECT —

Traces through data block links on a disk and reconstructs a valid BAM for the disk

CONCAT —

Appends one sequential file to the end of another

COPY —

Creates identical files

DCLOSE# —

Closes a working file

DIRECTORY —

Displays a list of file names on the disk

DLOAD —

Reads a program from disk

DOPEN# —

Opens a file for working

DSAVE —

Transfers program to disk

GET# —
Reads one byte from the disk

HEADER —
Formats new diskettes

INITIALIZE —

Inform DOS of disk change

INPUT# —

Read variables from the disk

MEMORY-EXECUTE —

Executes subroutines in DOS memory

MEMORY-READ —

Accesses byte pointed to by DOS memory address

MEMORY-WRITE —

Writes data to DOS memory

PRINT# —

Transfers data to open relative or sequential file

RECORD# —

Positions file pointer to any position in a file

RENAME —

Changes the name of an existing file

SCRATCH —

Destroys unwanted files

USER —

Provides a link to 6502 machine code within the disk unit

APPENDIX B. SUMMARY OF DOS ERROR MESSAGES

0	OK, no error exists.
1	Files scratched reponse. 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.
27	Checksum error in header
30	General syntax error.
31	Invalid command.
32	Long line.
33	Invalid file name.
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 read.
75	Format Speed Error
76	Controller Error.

APPENDIX C. PERMANENT ALTERATION OF DEVICE NUMBER

As assembled at the factory all CBM disk units have a device number of 8. This may be changed temporarily via the M—W command and will revert to 8 on power-up or reset. The device number may be changed permanently by means of modifications to printed circuit boards within the disk units. The hardware changes necessary differ for each model of disk unit.

WARNING

These hardware modifications should be performed only by qualified CBM service technicians. Alterations attempted by unauthorized personnel will void the warranty on your disk unit.

CHANGE 2031

Two diodes (CR18 and CR19) control device number on the 2031. The diodes are located adjacent to I.C. chip U3J on the digital PCB. To change the device number, cut the leads on one or both diodes as shown:

DEVICE NR.	CR18	CR19	
8	0	0	(0 = Unchanged)
9	0	1	(1 = Lead Cut)
10	1	0	
11	1	1	

CHANGE 4040 / 8050 / 8250

Three pins (22, 23, 24) on I.C. chip UE1(on the digital PCB) control device number on these units. These pins are normally strapped to ground by the circuit etch. Three small circular blocks appear just to the left of UE1- pin 22 is connected to the topmost of these blocks (when viewing the digital PCB from the front of the disk unit. To change device number either cut the appropriate trace(s) or remove UE1 and bend the correct pin(s) up so that they will not make connection when the chip is replaced.

DEVICE NR.	Pin 22	Pin 23	Pin 24	
8	0	0	0	(0 = Unchanged)
9	0	0	1	(1 = Cut/Bent)
10	0	1	0	
11	0	1	1	
12	1	0	0	
13	1	0	1	
14	1	1	0	
15	1	1	1	

CHANGE D9060 / D9090

Three pins (22, 23, 24) on I.C. chip 7G (on the topmost PCB) control the device number of these units. These pins are normally strapped to ground by the circuit etch. To change device number either cut the appropriate trace(s) or remove 7G and bend the correct pin(s) up so that they will not make connection when the chip is replaced.

DEVICE NR.	Pin 22	Pin 23	Pin 24	
8	0	0	0	(0 = Unchanged)
9	0	0	1	(1 = Cut/Bent)
10	0	1	0	
11	0	1	1	
12	1	0	0	
13	1	0	1	
14	1	1	0	
15	1	1	1	

APPENDIX D. COMMODORE DISK UNIT SPECIFICATIONS

MODEL	D9090	D9060	8250	8050	4040	2031
DRIVES	1	1	2	2	2	1
HEADS/DRIVE	6	4	2	1	1	1
STORAGE CAPACITY (Per Unit)						
Formatted	7.47 Mb	4.98 Mb	2.12 Mb	1.05 Mb	340 Kb	170 Kb
MAXIMUM (Each Drive)						
Sequential File	7.41 Mb	4.94 Mb	1.05 Mb	521 Kb	168 Kb	168 Kb
Relative File	7.35 Mb	4.90 Mb	1.04 Mb	183 Kb	167 Kb	167 Kb
Disk System Buffer RAM (Bytes)	4K	4K	4K	4K	4K	2K
DISK FORMATS (Each Drive)						
Cylinders (Tracks)	153	153	(77)	(77)	(35)	(35)
Sector/Cylinder	192	128	----	----	----	----
Sector/Track	32	32	23-29	23-29	17-21	17-21
Bytes/Sector	256	256	256	256	256	256
Free Blocks	29162	19442	4133	2052	664	664
TRANSFER RATES (Bytes/Sec)						
Internal	5 Mb	5 Mb	40 Kb	40 Kb	40 Kb	40 Kb
IEEE-488 Bus	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb
ACCESS TIME (Milli-seconds)						
Track-to-track	3	3	*	*	30	30
Average track	153	153	**	**	360	360
Average Latency	8.34	8.34	100	100	100	100
Speed (RPM)	3600	3600	300	300	300	300

* "gate" type = 30ms. "door" type = 5ms. Track-to-track.
** "gate" type = 750ms. "door" type = 125ms. Average track.

PHYSICAL DIMENSIONS						
Height (in.)	5.75	5.75	7.0	7.0	7.0	5.5
Width (in.)	8.25	8.25	15.0	15.0	15.0	8.0
Depth (in.)	15.25	15.25	13.75	13.75	13.75	14.25
Weight (lbs.)	21	21	28	28	28	20

ELECTRICAL SPECIFICATIONS

Voltages (ALL MODELS): 100-117, 220, or 240 VAC (European)
Frequency (ALL MODELS - HZ): 60 or 50 (European)

Power (Watts)	200	200	60	50	50	40

APPENDIX E ERROR IN RELATIVE RECORDS

An error exists in the Relative Records scheme that is used in all Commodore Disk Systems, 2031, 4040, 8050, 8250, 9060, 9090, and 8280. The error of itself will not corrupt any data files, nor will it cause errors in the disk system, however, if a particular program sequence (as described later) is being used to update files, the file will then be corrupted.

PROBLEM DESCRIPTION

Associated with Relative Files is a parameter defining the length of the records in the file. With the exception of record lengths 1, 2, 127 and 254, the records in the file will span or overlap the boundary between sectors on the disk.

In any group or series of even length records (10, 40, 66, etc.) that begins on a sector boundary (record #1, #128, #255, #509, etc.), the record that spans the 2nd and 3rd records in the series will be in error during reading when all of the following three conditions are true:

1. The total data field in the first record of the series (record 1, 128, 255, etc.) is shorter in length than the overlapped data (or spill) portion of the record which starts in the second sector and ends in the third sector. (e.g., for a record length of 100, record 6 will span sectors 2 and 3). The data field of record 6 will have all bytes in excess of byte 9 stored in sector 3. Thus, this condition will be met with a record length of 100 and a data field in record 6 that is 10 or more bytes longer than the data field of record one.

2. The program sequence that is running is a general update of records, where a record is read in, modified or examined and then written back out to the file. The following sequence will satisfy this condition:

```
1100  RECORD #1, (I): REM POINT TO ITH RECORD
1200  INPUT #1,A$: REM READ DATA IN
1300  (MODIFY OR EXAMINE A$)
1400  RECORD #1, (I): REM POINT TO RECORD AGAIN
1500  PRINT #1,A$: REM WRITE NEW (OR UNCHANGED) DATA
```

3. The records are being updated sequentially, beginning with a record that starts in the first sector

   eg: In the above example, adding lines:
   1000 for I=2 to 10
   1600 NEXT

   will satisfy this condition.

Under the above conditions, the data sent to the computer for record 6 will be truncated to the extent that the data overlap of record 6 into sector 3 exceeds the length of the data in record one. The following program will demonstrate the bug:

```
100    A$ = "1234"
200    B$ = "123456789ABCDE"
300    DOPEN#1, "TEST", L100
400    PRINT#1, A$: IF DS THEN STOP
500    FOR I = 2 TO 10
600    PRINT#1, B$: IF DS THEN STOP
700    NEXT: DCLOSE#1
800    DOPEN#1, "TEST":B  $="123456789ABCDE"
900    FOR I=2 TO 10
1000   RECORD#1, (I): IF DS THEN STOP
1100   INPUT#1, C$: IF DS THEN STOP
1200   IF LEN(C$) <> LEN(b$) THEN PRINT "REC #"I; B$; LEN(B$); C$; LEN(C$):GOTO 1500
1300   RECORD#1, (I): IF DS THEN STOP
1400   PRINT#1, B$: IF DS THEN STOP
1500   NEXT
1600   DCLOSE#1
```

Running the above will produce a printout of:

REC #6 123456789ABCDE 14 123456789ABCD 13

The data in record 6 on the disk is not corrupted, as can be shown by running the program again, starting from line 800. The error is occurring in the DOS RAM because an end of data pointer is not being set correctly. However, if a blind read, update, rewrite is being followed then the truncated data from record 6 will be operated on and rewritten, resulting in corruption of the file.

Note also that the truncation is not detectable by error monitoring as no read error occurs.

PROBLEM RESOLUTION

Since the error can occur with the general conditions as described, it is not possible to detect any unique conditions, however there does exist a simple programming technique to bypass the error. Adding to the example program the line:

```
1450 RECORD#1, (I)
```

will cause the DOS to reset the pointer for the sensitive record, eliminating the problem.

The addition of this statement immediately after the print statement will not materially affect the run time of a program as the DOS will not have to reaccess the sector.

INDEX

A

B

C

D

E

F

ADDENDUM TO DISK SYSTEM USER REFERENCE GUIDE

The Commodore SFD 1001, single disk drive unit operates, for the most part, the same way as the Commodore 8250 dual disk drive.

To learn the basic operations available in your SFD 1001 disk drive, please refer to the 8250 section of your Disk System User Reference Guide.

However, because the 8250 has two drives and the SFD is only one drive the information presented below defines and explains all the modifications you'll have to make to the 8250 section of your Disk System User Reference Guide.

1) PAGE 45: RELATIVE FILES IN 8250 DISK UNITS
In this section, all references to 8250 disk units also apply to the SFD 1001.

2) PAGE 45: USING 8050 DISKS IN 8250 DRIVES
References to the 8250 disk unit, also apply to the SFD 1001 disk unit with the following exception: You can NOT back-up or copy 8050 disks on the SFD 1001 because it's a single unit drive, not a dual unit drive.

3) PAGE 48: DOS 2.5
Your manual refers to the use of DOS 2.5 in 8050 disk drives. The Disk Operating System has been upgraded so that DOS 2.7 is used in 8050, 8250, and SFD 1001 disk drives.

4) PAGE 57: MEMORY UTILITIES – (Memory-write)
When two or more disk units are attached to the computer, the device number of each unit must be made unique or neither unit will operate correctly. References on page 56 for altering the device number on 8250 disk units, also apply to the SFD 1001 disk drive unit.

5) PAGES 66–67: DISK STORAGE FORMATS
Information provided on pages 66–67 also apply to the SFD 1001 disk drive unit.

G

H

I

J

M

N

O

P

R

S

U

commodore COMPUTER

Part Number 320972-01