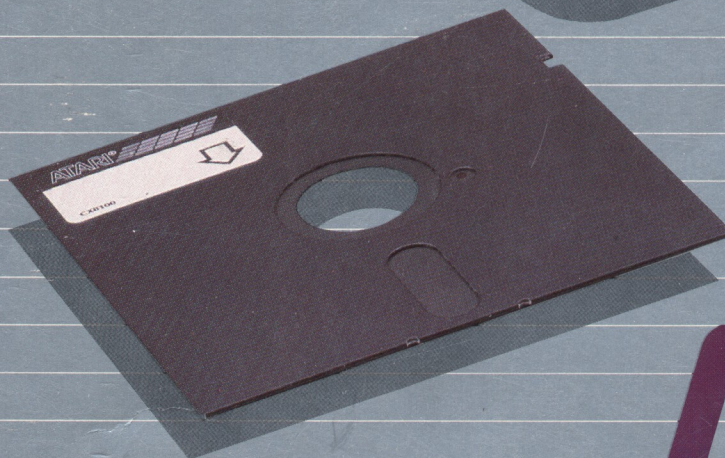
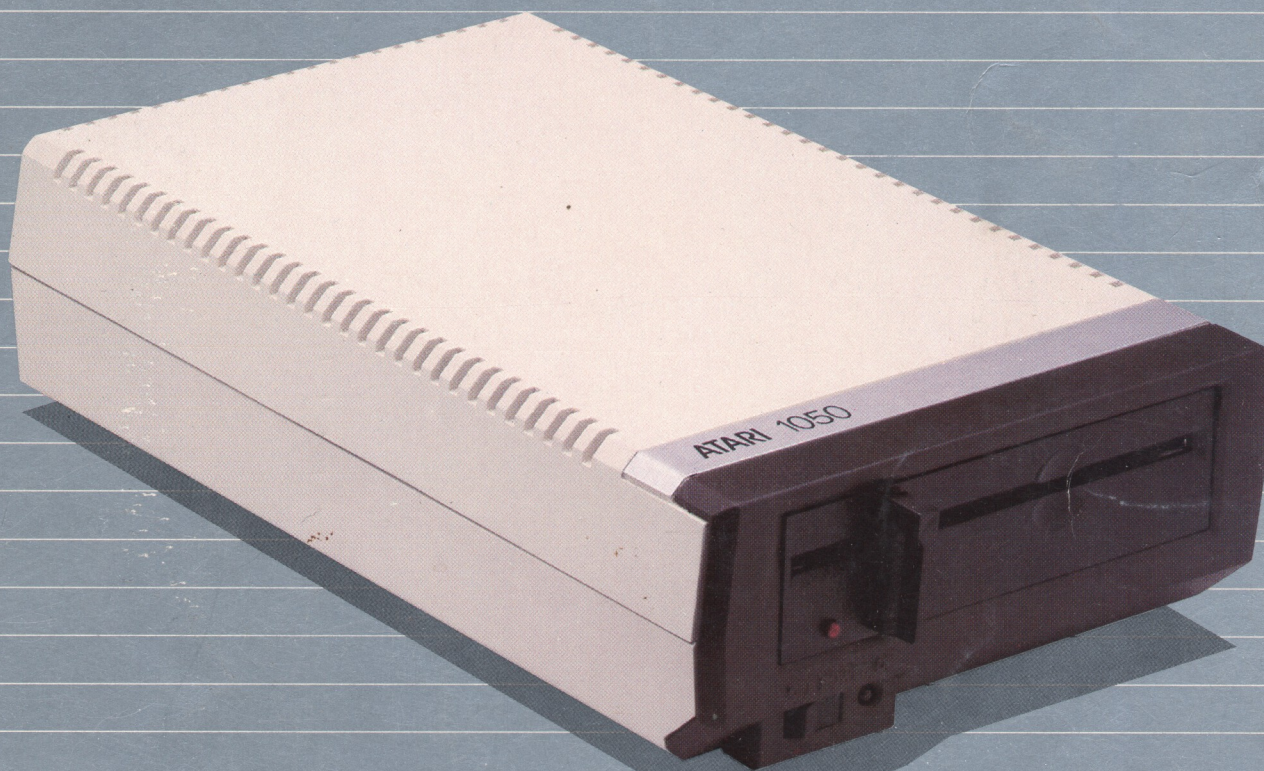


ATARI® 1050™ DISK DRIVE

DISK OPERATING SYSTEM II
REFERENCE MANUAL



ATARI
HOME COMPUTERS

**NOTE TO
ATARI® 1050™
DISK DRIVE
OWNERS:**

References to the ATARI 810™ Disk Drive in this *Disk Operating System II Reference Manual* are directly applicable to the ATARI 1050 Disk Drive. When the *Reference Manual* refers you to the ATARI 810 *Owner's Guide*, see the *Owner's Guide* that came with your ATARI 1050 Disk Drive.

DOS II and the ATARI 1050 Disk Drive work with the new ATARI 1200XL™ Home Computer as well as the ATARI 400™ or ATARI 800™ Home Computers.

Please disregard all references to the ATARI 815™ Dual Disk Drive.

DISK OPERATING SYSTEM II REFERENCE MANUAL



A Warner Communications Company 

Every effort has been made to ensure that this manual accurately documents this product of the ATARI Computer Division. However, because of the ongoing improvement and update of the computer software and hardware, ATARI, INC. cannot guarantee the accuracy of printed material after the date of publication and cannot accept responsibility for errors or omissions.

Reproduction is forbidden without the specific written permission of ATARI, INC., Sunnyvale, CA 94086. No right to reproduce this document, nor the subject matter thereof, is granted unless by written agreement with, or written permission from the Corporation.

PRINTED IN SINGAPORE

MANUAL AND PROGRAM CONTENTS ©1981 ATARI, INC.

PREFACE

The ATARI® *Disk Operating System II (DOS II) Manual* has been structured for two levels of users. The new user will find DOS II explained in well-defined terminology and sequential operating procedures. However, the new system user should make the following preparations:

- Read the *ATARI 400™ or ATARI 800™ Computer Operator's Manual*.
- Read the *ATARI 810™ Disk Drive or ATARI 815™ Dual Disk Drive Operating Manual*.
- Acquire some familiarity with an ATARI programming language (preferably BASIC).

As the new user becomes familiar with DOS II in Sections 1 through 4, he or she may then explore Sections 5 and 6, which are designed for the more experienced user.

The more experienced user is one who has worked with ATARI DOS I (9/24/79 version) or has prior knowledge of the ATARI Personal Computer System, ATARI BASIC, and possibly assembly and other programming languages.

For both levels, explanations are followed by graphics whenever possible. Our intention is to present the material in the clearest, most concise way possible for users at each level.

Notes and terminology used with DOS II can be found in Appendix B; a glossary of terms is available in Appendix J; and the use of BASIC commands with DOS II is detailed in Section 5.

The experienced user should read Appendix I which explains the major differences between DOS I and DOS II, especially noting the addition of the MEM.SAV function.

CONTENTS

PREFACE	v
---------	---

HOW TO USE THIS MANUAL	xi
------------------------	----

1	GETTING STARTED WITH DOS II	1
---	-----------------------------	---

	Setting Up	1
	Adding More Disk Drives	2
	Setting Drive Codes	2
	Labeling Disk Drives	3
	Inserting a Diskette	3
	What Is a DOS Menu?	4
	How Do You Call Up a DOS Menu?	4
	DOS II Menu Options	5

2	DISKETTES	9
---	-----------	---

	The Master Diskette	9
	DOS.SYS File	9
	DUP.SYS File	9
	AUTORUN.SYS File	9
	ATARI 810 Formatted Diskettes II	9
	How To Format a Diskette	10
	Making a System Diskette From Your	
	Master Diskette	12
	Creating a System Diskette With the ATARI	
	810™ Disk Drive	12
	Creating a System Diskette Using an ATARI	
	815™ Dual Disk Drive	13
	Write-Protecting Your Diskettes	14
	Labeling Diskettes	15
	Single-Density and Double-Density Diskette	
	Recording	15
	Which Diskettes To Use	16
	How To Store Diskettes	16

3	USING DOS II	19
---	--------------	----

	Identifying Your Diskette Files	19
	Filename Extenders and Their Use	19
	Wild Cards	21
	Boot Errors	22
	Saving, Loading, and Running Programs	23

4	SELECTING A DOS MENU OPTION	27
A.	Disk Directory	27
	Parameters for the Disk Directory Option	28
B.	Run Cartridge	29
C.	Copy File	30
D.	Delete File	32
E.	Rename File	32
F.	Lock File	35
G.	Unlock File	35
H.	Write DOS File	35
I.	Format Diskette	36
J.	Duplicate Disk	37
	Duplication Using a Single Disk Drive	37
	Duplication Using Multiple Disk Drives	38
K.	Binary Save	39
	Advanced User Information About Optional Parameters	39
	Using Binary Save With Optional Parameters	40
	Structure of a Compound File	41
L.	Binary Load	43
M.	Run at Address	44
N.	Create MEM.SAV	44
	Why Have a MEM.SAV File?	45
	Using MEM.SAV to Write Assembly Language Programs	45
	Using MEM.SAV to Load Binary Files	46
O.	Duplicate File	47
5	MORE USER INFORMATION	49
	BASIC Commands Used With DOS	49
	Tokenized and Untokenized Files	49
	LOAD	49
	SAVE	50
	LIST	50
	ENTER	50
	RUN	51
	Input/Output Control Blocks	51
	IOCB's With Input/Output Commands	51
	Using OPEN/CLOSE Commands	51
	Using INPUT/PRINT Commands	53

Direct Accessing With the NOTE/POINT Commands	54
Using the PUT/GET Commands	57
Using the STATUS Command	58
Substituting the XIO Command for DOS Menu Options	60
Saving and Loading Programs and Data With ATARI BASIC	62
LIST and ENTER	63
OPEN and CLOSE	66
Accessing Damaged Files	67
The AUTORUN.SYS File	67
<hr/>	
6 ADDITIONAL INFORMATION ABOUT THE DISK DRIVE SYSTEM	69
<hr/>	
ATARI Diskettes	69
ATARI Disk Drive	69
ATARI 815 Disk Drive	69
Disk Drive Operation	70
<hr/>	
APPENDICES	71
<hr/>	
A Alphabetic Directory of Basic Words Used With Disk Operations	71
B Notes and Terminology Used With DOS II	73
C Basic Error Messages and How To Recover	75
D DOS II Memory Map For 32K Ram System	81
E Hexadecimal-to-Decimal Conversion Table	83
F How To Speed Up Data Transfers to Disk Drive	85
G How To Increase User RAM Space	87
H Major Differences Between DOS I (9/24/79) and DOS II	89
I Structure of a Compound File	91
J Glossary of Terms	93
<hr/>	
INDEX	99
<hr/>	
ILLUSTRATIONS	
<hr/>	
1-1 Disk Drive Configurations Available	1
1-2 Drive Code Settings	2
1-3 Inserting a Diskette Into a Disk Drive	3
1-4 The DOS II Menu	4

2-1	A Formatted Diskette	10
2-2	Write-Protecting a Diskette	15
2-3	Correct Diskettes for Your Disk Drive	16
3-1	Structure of a Filespec	20
3-2	Examples of Legal and Illegal Filenames	20
3-3	DOS Menu Options That Can and Cannot Use Use Wild Cards	21
3-4	Boot Errors	22
3-5	Sample Interest Program	24
4-1	Using the Disk Directory Option	29
4-2	Using the Copy File Option	32
4-3	Using the Delete File Option	33
4-4	Using the Rename File Option	34
4-5	Using the Lock File Option	35
4-6	Using the Unlock File Option	35
4-7	Using the Write DOS File Option	36
4-8	Using the Format Disk Option	36
4-9	Using the Duplicate Disk Option With Single Disk Drive	38
4-10	Using the Duplicate Disk Option With Dual or Multiple Disk Drives	38
4-11	The Most Elementary Use of Binary Save	39
4-12	Six-Byte Header Table for Binary Save	39
4-13	Using Binary Save With Optional Parameters	40
4-14	Using Binary Save to Save Compound Files	41
4-15	Converting an Existing Load-Only File to a Load-and-Go File	43
4-16	Using the Binary Load Option	43
4-17	Using the Run at Address Option	44
4-18	Creating a MEM.SAV File	44
4-19	Example of MEM.SAV Usage	45
4-20	Using the Duplicate File Option	48
5-1	Example of Program Chaining	50
5-2	Explanation of OPEN Statement Parameters	52
5-3	Example of Opening and Closing a File	52
5-4	Sample INPUT/PRINT Program	53
5-5	Sample NOTE Program	54
5-6	Sample Run of NOTE Program	55
5-7	Sample POINT Program	56
5-8	Sample Run of POINT Program	57
5-9	Sample PUT Program	57
5-10	Sample GET Program	58
5-11	Sample Run of PUT/GET Program	58
5-12	Sample STATUS Program	60

5-13	Sample XIO Program	62
5-14	Sample Interest Program	64
5-15	Sample Run of Interest Program	64
5-16	Sample Program to Create a Data File	65
5-17	Run of Sample Data File	66
5-18	The Information Stored on Diskette	66
5-19	Get Byte Program	67
5-20	An AUTORUN.SYS Example for the Advanced User	68

HOW TO USE THIS MANUAL

This manual has been developed with the user in mind. Each section of information represents one phase of the second version of the ATARI Disk Operating System (DOS II). The newcomer to DOS can easily find the information needed to get started with DOS II without feeling encumbered by extraneous information. The experienced user, however, can quickly find and use the data required to perform more complex operations.

FOR THE NEW DOS USER

Section 1 explains how to use this manual and the procedures for the most elementary operations:

- Definition of DOS
- Setting up the system
- Explaining the DOS Menu

Sections 2 and 3 discuss and explain everything about your diskettes from formatting to storage. The novice to DOS should read through Sections 1 and 2 before sitting down to work. This will give you an opportunity to become familiar with what you are to do.

FOR THE NOVICE AND MORE EXPERIENCED USER

Section 3 starts your involvement with DOS II, explaining how to identify your diskette files using filenames and filename extenders; this section also introduces you to Loading and Saving programs.

Section 4 is the crossover point between the new and more experienced user, and has a detailed description of each DOS Menu option and how to use it. Some of these options will only be of interest to users who are familiar with the Assembler Editor cartridge and the hexadecimal system.

FOR THE MORE EXPERIENCED USER

Section 5 reviews the BASIC commands used with DOS II and gives sample programs showing the I/O commands in actual use. Each command format is also followed by an example showing the types of data going into each parameter. Section 6 contains further information about ATARI Disk Drives and diskettes (which may be of interest to both levels of users), along with details about storing and retrieving data.

The balance of the manual contains a glossary of terms and additional information for the advanced user such as:

- Memory maps
- Errors
- Saving RAM space

GETTING STARTED WITH DOS II

DOS (pronounced doss) is an acronym for Disk Operating System. Without a DOS, your ATARI® Personal Computer System cannot communicate with your ATARI 810™ Disk Drive. The DOS consists of comprehensive utility routines that allow you to:

- Store programs on diskette
- Retrieve programs from diskette
- Create and add to data files needed by programs
- Make copies of disk files
- Delete old files from a diskette
- Load and save binary files (for the advanced user)
- Move files to and from memory, the screen, diskette, and printer.

If you have never used a DOS before, you will find the ATARI DOS II simple to understand and easy to use. If you previously used the ATARI DOS I, you will find differences in loading DOS II and changes to both the menu options and their command parameters. These changes make DOS II more advantageous because you have more available user memory space and greater flexibility than with the 9/24/79 version of DOS I (see Appendix H).

SETTING UP

To start, you must attach your ATARI 810 Disk Drive or ATARI 815™ Dual Disk Drive to your ATARI 400™ or ATARI 800™ Personal Computer System, making sure your computer has at least 16K Random Access Memory (RAM). For complete instructions on setting up your equipment, please refer to your *ATARI 400* or *ATARI 800 Operator's Manual*. The procedures for attaching the ATARI 815 Dual Disk Drive and the ATARI 810 Disk Drive to your ATARI Personal Computer System are contained in the respective Disk Drive Operator's Manuals.

ADDING MORE DISK DRIVES

Should you want to attach additional ATARI Disk Drives to your personal computer system, you do so by "daisy-chaining." In the back of each ATARI Disk Drive are two outlets (labeled I/O PERIPHERAL); attach the I/O cord from your second disk drive to Drive 1. Then attach the other cord from Drive 1 to the computer console. Figure 1-1 shows you some of the different disk drive configurations available.

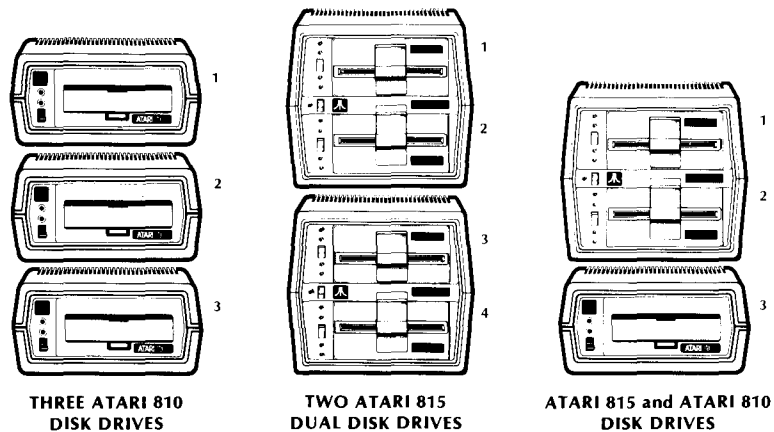
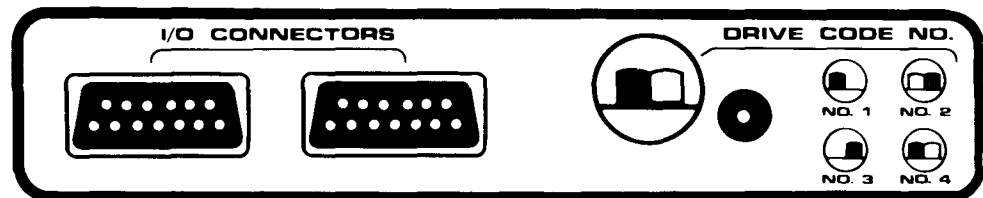


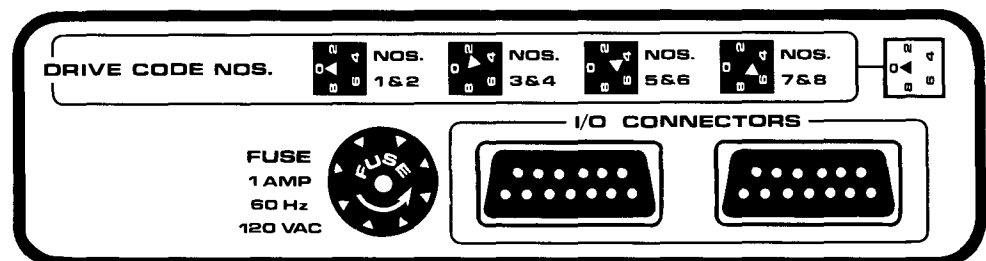
Figure 1-1 Disk Drive Configurations Available

SETTING DRIVE CODES

Looking at the back of your ATARI 810 Disk Drive(s), you will see a hole with two tabs. The white and black tabs must be moved on each drive according to that drive's designation. Refer to Figure 1-2a for the proper drive code setting. The back of your ATARI 815 Dual Disk Drive has a dial you must rotate to the proper position to set the drive code setting (see Figure 1-2b). If you have only one ATARI 815 Dual Disk Drive, the top drive is always set to 1 and the bottom drive set to 2. It is important to make sure that each drive has a separate drive code setting.



(a) ATARI 810 Disk Drive



(b) ATARI 815 Dual Disk Drive

Figure 1-2 Drive Code Settings

LABELING YOUR DISK DRIVES

Once you have properly set the drive codes, label each disk drive with its appropriate number so you will not make a mistake when using disk drive functions. It is imperative that your Master or System Diskette ALWAYS be placed in Disk Drive 1.

INSERTING A DISKETTE

Inserting a diskette into an ATARI Disk Drive is a simple, but very important procedure. If the diskette is improperly positioned, it can cause boot (starting-up) errors during DOS loading procedures, and can also damage the diskette.

Turn on the disk drive(s), and wait for the BUSY light to go off. Insert the diskette as follows:

1. Remove the diskette from its protective paper sleeve.

Caution: Hold the diskette ONLY by its black, sealed envelope. DO NOT touch any exposed surfaces of the diskette, as this will impair or destroy its read/write capabilities. DO NOT hold the diskette by placing your fingers through the center hole. DO NOT try to remove the diskette from its black, sealed envelope.

2. Hold the diskette so the labeled side is up, with the label toward you, and the arrow on the label is pointing toward the disk drive door (see Figure 1-3). Also, if your diskette has a write-protect notch, this should be on your left. Please note that your Master Diskette does not have a write-protect notch because it is automatically write-protected at the factory.

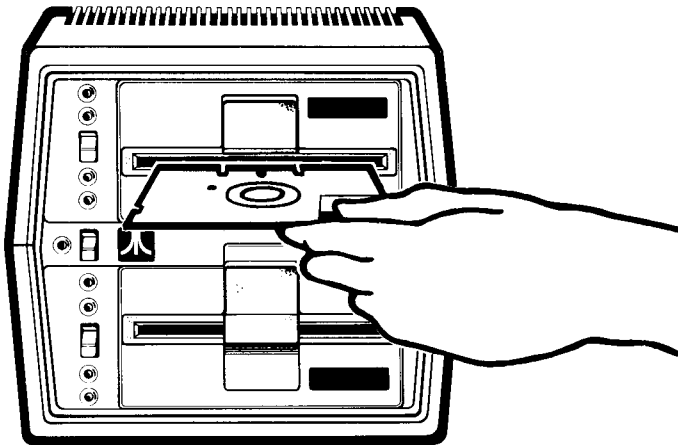


Figure 1-3 Inserting a Diskette Into a Disk Drive

3. Open the door to the disk drive (upper drive if you have an ATARI 815 Dual Disk Drive) and gently but firmly slide in the diskette.
4. Close the disk drive door.

WHAT IS A DOS MENU?

The DOS Menu is loaded into your computer from your Master Diskette, which is always inserted into Disk Drive 1. The DOS Menu is similar to a restaurant menu; a selection of applications is presented to you via your television screen. You make your selection by typing the appropriate code letter and pressing **RETURN**. Your selected application is now available for use.

The DOS II Menu selections for both the ATARI 810 Disk Drive (single density—128 bytes per sector) and the ATARI 815 Dual Disk Drive (double density—256 bytes per sector) are identical. The only difference is the version designation on the screen as explained below:

- Single Density (Used on the ATARI 810 Disk Drive): You will see a Version 2.0S in the upper right corner of your screen. The S stands for single density.
- Double Density (Used on the ATARI 815 Dual Disk Drive): You will see a Version 2.0D in the upper right corner of your screen. The D stands for double density.

HOW DO YOU CALL UP A DOS MENU?

After you have your television set turned on, your Master Diskette inserted into the disk drive, and your disk drive turned on, you are ready to load DOS II into the computer memory as follows:

With a Cartridge Installed

When you turn your computer on:

1. The BUSY light on the disk drive will go on during the loading process. DO NOT attempt to remove the diskette while this light is on. If you have inserted the ATARI BASIC cartridge, a READY prompt will appear on the screen once DOS II is loaded. (If you have inserted the Assembler Editor cartridge, the prompt will be EDIT.) This completes the first part of the loading procedure.
2. Type **DOS** and press **RETURN**. The DOS II Menu will display on the screen (see Figure 1-4). This completes the second part of the loading procedure.

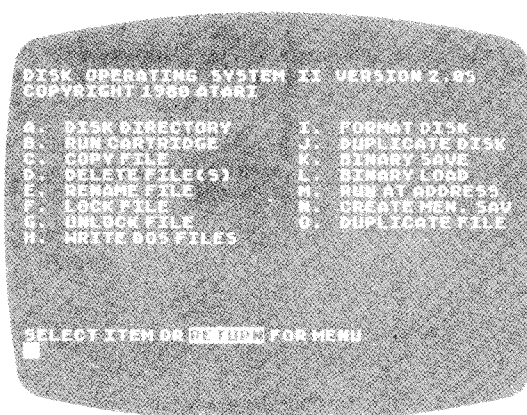


Figure 1-4 The DOS II Menu

With a cartridge present, the first part of the load brings in the FILE MANAGEMENT SUBSYSTEM and MINI-DOS. The second part brings in the DISK UTILITY PACKAGE and the DOS Menu.

With No Cartridge Installed

When you turn your computer on, DOS II will load entirely without user interaction. The BUSY light on the disk drive will go on during the loading process. DO NOT attempt to remove the diskette while this light is on. The DOS Menu will be displayed automatically once DOS II is loaded.

After you make a code letter selection, a prompt message will display on the screen requesting information from you. This capability makes it a “self-prompting” menu. The prompt message that appears most frequently reads:

SELECT ITEM OR RETURN FOR MENU

This means the system waits for you to do one of the following:

- Type in one of the alphabetic letters and press **RETURN** to bring up your selection, or
- Press **RETURN**, which redisplay the DOS Menu.

THE DOS II MENU OPTIONS

Below are the DOS II Menu options and a brief explanation of what they do. You are advised NOT to use these options until you understand them thoroughly (a detailed explanation appears in Section 4).

DISK DIRECTORY

This contains a list of all the files on a diskette. If you select option **A** and press **RETURN** TWICE, you can display the filenames, extenders (if any), the number of sectors allocated to the file, and the number of free sectors still available.

RUN CARTRIDGE

(Can ONLY be used with a cartridge installed in the computer console.) This allows you to return control of your system to the cartridge inserted in the cartridge slot (the left cartridge slot in the ATARI 800 Personal Computer System).

COPY FILE

Use this option when you have two or more disk drives and you want to copy files from one diskette to another. Also use this option if you want two files with the same information on the same diskette by assigning a second name to the original file.

DELETE FILE

This option lets you erase a file from the diskette, increasing your available sector space.

RENAME FILE:

Use this option when you want to change the name of a file.

LOCK FILE

This option prevents you from changing, renaming, or accidentally erasing the file. You will still be able to read the file, but will not be able to write to it. When the directory is displayed, an asterisk is placed in front of the file name to indicate it is locked.

UNLOCK FILE

This removes the asterisk from in front of the file name and allows you to make changes to the file, rename it, or delete it.

WRITE NEW DOS

Use this option to replace or add the DOS files (DOS. SYS and DUP. SYS) on your Master Diskette onto a diskette in any disk drive.

FORMAT DISKETTE

This option is used to format a blank diskette, which is necessary before you can write anything onto it. Be sure you do not have any files you want to keep on the diskette before formatting. This option is not normally used on ATARI 810 Formatted Diskettes (CX8111).

DUPLICATE DISK

This is the option you choose when you want to create an exact duplicate of a diskette (for more detailed information, refer to Section 2 on Backing Up Diskettes).

DUPLICATE FILE

This option enables you to copy a file from one diskette to another, even if you only have a single disk drive. (See the section on Duplicating Files for more detail.)

CREATE MEM.SAV

This option allows you to create available sector space on the diskette for the program in RAM to be stored while the DUP. SYS file is being used. (For more information, see CREATE MEM.SAV in Section 4.) We advise you to create a MEM.SAV file on each new diskette you intend to use as a System Diskette. As you become more familiar with the DOS, you may find there are cases where a MEM.SAV file serves no useful function. Hence, the inconvenience of waiting for the file to load into memory may warrant deleting it from the diskette. An example would be when there is no program in RAM that you want to protect when typing in DOS.

BINARY SAVE*

With this option you can save the contents of specified memory locations on a diskette. (Manipulates assembly language programs.)

BINARY LOAD*

This option lets you retrieve an object file from a diskette. It is the reverse function of BINARY SAVE. (Manipulates assembly language programs.)

RUN AT ADDRESS*

With this option you can enter the hexadecimal starting address of an object program after it has been loaded into RAM with a BINARY LOAD. (Executes assembly language programs.)

***Note:** BINARY SAVE, BINARY LOAD, and RUN AT ADDRESS are for the advanced user of DOS II and are explained in greater detail in Section 4.

DISKETTES

THE MASTER DISKETTE

The Master Diskette contains the disk operating system programs. These programs include all the system file management and utility routines necessary to make your disk drive function with your ATARI Personal Computer System. Without a disk operating system, you cannot access the disk drive.

Each Master Diskette contains the following files:

DOS.SYS File

DOS.SYS is a file containing the File Management Subsystem (FMS) and the RAM-resident portion of the DUP.SYS (frequently referred to as "mini-DOS"). The RAM-resident portion of DUP.SYS contains the subfunctions that can be controlled by the FMS: DELETE FILE, RENAME FILE, LOCK FILE, UNLOCK FILE, and FORMAT DISK.

DUP.SYS File

This is the Disk Utility Package that contains the DOS Menu and DOS subfunctions NOT controlled by the FMS. Whenever you want to see the DOS Menu or perform these DOS subfunctions (BINARY LOAD, BINARY SAVE, RUN AT ADDRESS, RUN CARTRIDGE, COPY FILE, DUPLICATE FILE, and DUPLICATE DISK), you must load the DUP.SYS file into RAM by typing **DOS** and pressing **RETURN**.

Note: Normally when you bring the DUP.SYS file into RAM, it writes over data in the lower program area occupied by BASIC or assembly language programs. However, when you create a MEM.SAV file (see section on MEM.SAV) on your diskette, the mini-DOS saves any data in RAM to diskette before loading DUP.SYS. When you are finished using the DUP.SYS functions, MEM.SAV allows you to reload your program automatically.

AUTORUN.SYS File

This file is used to poll (check) the peripheral units (if any) attached to your ATARI Personal Computer System and to run machine code programs (see Section 4 of this manual for more detail on AUTORUN.SYS).

ATARI 810 FORMATTED DISKETTES II

In addition to the Master Diskette II (CX8104), your ATARI 810 Disk Drive comes with an ATARI 810 Formatted Diskette II (CX8111). Although this diskette has no files or program data on it, it has been preformatted at the factory. Preformatting means the diskette was divided into tracks and sectors before packaging (see Figure 2-1) so that it has an improved sector layout. This improved sector layout makes it possible for you to store and retrieve information more rapidly than is possible with diskettes formatted on your ATARI 810 Disk Drive. This "empty" diskette is provided so you can make a backup copy of your Master Diskette. This backup

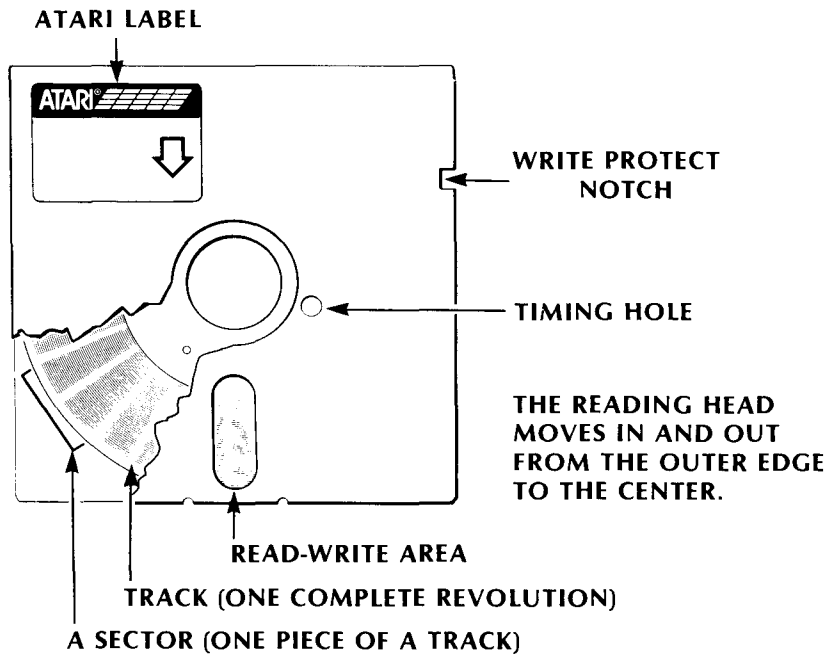


Figure 2-1 A Formatted Diskette

copy, called a System Diskette, is the one you will actually work with, ensuring the safety of your original Master Diskette (see Making a System Diskette).

If, at any time, you decide to erase the files from a preformatted diskette, use the DOS Menu selection D. DELETE FILE(S). This will save the improved sector layout that allows the diskette to run faster. Should you reformat the diskette for any reason (perhaps to test for possible bad sectors), it will no longer have the improved sector layout.

Note: You can also use blank diskettes that have not been preformatted at the factory with the ATARI 810 Disk Drive. You will have to format these diskettes yourself using the FORMAT DISK menu option before you can write DOS FILE or store programs on them (see How to Format a Diskette). However, you will not have the advantage of the improved speed of the preformatted diskettes.

HOW TO FORMAT A DISKETTE

You will need to format any blank diskette before you can write on it (unless you are using an ATARI 810 Formatted Diskette II (CX8111)). Formatting organizes a diskette so DOS II will know where information is located. Unlike a phonograph record that has visible spiraling grooves imprinted onto it, the diskette has magnetically inscribed grooves that are not visible.

Once you have formatted a diskette, it will contain 40 of these concentric tracks, which are divided into 18 pie-shaped wedges called sectors. You ascertain the storage capacity by multiplying the number of tracks (40) by the number of sectors per track (18), which gives you 720 sectors. However, 13 of the 720 sectors are used

by DOS II and are not available to you for writing files and data. The actual breakdown is as follows:

- 3 sectors used for booting the system
- 8 sectors used for the Directory
- 1 sector used for the Volume Table of Contents
- 1 Sector 720 is not addressable
- 13 Total

As a result, you actually have a total of 707 sectors to which you may write data. The ATARI 810, a single-density disk drive, can store 128 bytes of information on each sector of the diskette. Because 3 bytes per sector are allocated for the File Management Subsystem, the total storage capacity per single-density diskette is 88,375 bytes.

To format a diskette, you must use the I. FORMAT DISK option on the DOS Menu. This is not necessary if you are using ATARI 810 Formatted Diskettes II (CX8111). If you are using an ATARI 810 Disk Drive as Drive 1, see instructions (A) below. If you are using an ATARI 815 Dual Disk Drive as Drives 1 and 2, see instructions (B).

(A) Using the ATARI 810 Disk Drive:

1. Turn the disk drive on. Wait for the BUSY light to go off.
2. Make sure the switch on the back of the disk drive is set to No. 1 (refer to Figure 1-2 for drive code settings).
3. Insert the ATARI 810 Master Diskette II (CX8104) and close the drive door.
4. Turn the computer console on. DOS will load into the computer memory.
5. When the READY prompt appears (if you have inserted the ATARI BASIC cartridge), type **DOS** and press **RETURN**. After a few seconds the DOS Menu will appear on the screen. (If no cartridge is inserted, the DOS Menu will appear on the screen automatically.)
6. Type **I** for the FORMAT option and press **RETURN**.
7. When the prompt message WHICH DRIVE TO FORMAT appears, remove the Master Diskette from the disk drive and insert a BLANK diskette. Close the door, type **1** and press **RETURN**.
8. When the prompt message TYPE Y TO FORMAT DISK 1 appears, type **Y** and press **RETURN**. The BUSY light will come on and the system will format the diskette.
9. When the prompt message SELECT ITEM OR RETURN FOR MENU appears, the formatting is complete, and you can write files to that diskette.

If you have two or more ATARI 810 Disk Drives, you can format a blank diskette on any drive. However, you must know the drive code setting of the drive you use, so you can respond to the prompt WHICH DRIVE TO FORMAT.

(B) Using the ATARI 815 Dual Disk Drive:

1. Turn the disk drive(s) on. Wait for the BUSY light to go off.

-
2. Make sure the switch on the back of the disk drive is set properly (refer to Figure 1-2 for drive code settings).
 3. Insert the ATARI 815 Master Diskette II (CX8201) into Drive 1 and close the drive door.
 4. Turn the computer console on. DOS will load into the computer memory.
 5. When the READY prompt appears (if you have inserted the ATARI BASIC cartridge), type **DOS** and press **RETURN**. After a few seconds, the DOS Menu will appear on the screen. (If no cartridge is inserted, the DOS Menu will appear on the screen automatically.)
 6. Type **I** for the FORMAT option and press **RETURN**.
 7. When the prompt message WHICH DRIVE TO FORMAT appears, place the blank diskette into Drive 2, close the door, type **2**, and press **RETURN**.
 8. When the prompt message TYPE Y TO FORMAT DISK 2 appears, type **Y** and press **RETURN**. The BUSY light will come on and the system will format the diskette in Drive 2.
 9. When the prompt message SELECT ITEM OR RETURN FOR MENU appears, the formatting is complete, and you can write files to that diskette.

MAKING A SYSTEM DISKETTE FROM YOUR MASTER DISKETTE

The first disk operation you need to perform is duplicating your Master Diskette. This is done to protect your Master Diskette from any inadvertent damage. The duplicate of your Master Diskette is referred to as the System Diskette (working copy), and is the one you will normally use to load DOS into RAM.

You create the System Diskette in one of two ways, depending on which disk drive you have.

CREATING A SYSTEM DISKETTE WITH THE ATARI 810 DISK DRIVE

1. Turn the television set and disk drive on and wait for the BUSY light (top red light) to go off.
2. Remove the Master Diskette from its protective paper sleeve.
3. Insert the Master Diskette into the disk drive and close the drive door.
4. Turn the computer console on.
5. Assuming you have a BASIC cartridge inserted in the console, you will see a READY prompt message. Type **DOS** and press **RETURN**. (If NO cartridge is inserted in the console, the DOS Menu will appear on the screen automatically.)
6. Remove the Master Diskette and insert a formatted diskette into the disk drive. This empty diskette can be one of the following:
 - An ATARI 810 Formatted Diskette II (CX8111)
 - A diskette you have previously formatted using DOS II
 - A diskette you have reformatted using DOS II

-
7. Type **H** and press **RETURN** for the WRITE DOS FILES option.
 8. When the prompt message DRIVE TO WRITE DOS FILES TO? appears, type **1** and press **RETURN**.
 9. When the prompt message TYPE Y TO WRITE DOS TO DRIVE 1 appears, type **Y** and press **RETURN**.
 10. The message WRITING NEW DOS FILES will appear on the screen.
 11. When the prompt message SELECT ITEM OR RETURN FOR MENU appears, the Master Diskette has been duplicated and you have created a System Diskette.

Note: At this point we strongly recommend you create a MEM.SAV file (see subsection on MEM.SAV later in this section). MEM.SAV allocates a specified number of sectors on the System Diskette (or any diskette) for storing the resident RAM program while you are using the DOS functions; i.e., the DUP.SYS file (see section explaining the DUP.SYS function).

12. Type **N** and press **RETURN** to create a MEM. SAV file on your System Diskette.
13. When the prompt message TYPE Y TO CREATE MEM. SAV appears, type **Y** and press **RETURN**.
14. When the prompt message SELECT ITEM OR RETURN TO MENU appears, your System Diskette will have a MEM. SAV file on it.

If you have two or more ATARI 810 Disk Drives, you can insert the formatted diskette into any drive before choosing Menu Option H. WRITE DOS FILES. However, you must remember which drive you are using so you will be able to answer the prompt, DRIVE TO WRITE DOS FILES TO?, in step 8 above.

CREATING A SYSTEM DISKETTE USING THE ATARI 815 DUAL DISK DRIVE

1. Turn on the ATARI 815 Dual Disk Drive. Make sure the drive code on the back of the drive has been set properly (see Figure 1-2). The upper drive should be labeled as Drive 1 and the lower drive as Drive 2.
2. Remove the Master Diskette from its protective paper sleeve.
3. Insert the Master Diskette into Drive 1 and close the drive door.
4. Turn the computer console on.
5. Assuming you have inserted a BASIC cartridge, you will see a READY prompt message on the screen. Type **DOS** and press **RETURN**. (If no cartridge is inserted in the console, the DOS Menu will appear automatically on the screen.)
6. When the prompt message SELECT ITEM OR RETURN FOR MENU appears, type **H** and press **RETURN** to select the WRITE DOS FILES option.
7. When the prompt message DRIVE TO WRITE FILES TO appears, place a formatted diskette into Drive 2 and close the drive door. Type **2** and press **RETURN**.

-
8. When the prompt message TYPE Y TO WRITE DOS TO DRIVE 2 appears, type **Y** and press **RETURN**.
 9. The message WRITING NEW DOS FILES appears on the screen.
 10. When the DOS Menu and prompt message SELECT ITEM OR RETURN FOR MENU appears, you have created a System Diskette.
Note: At this point we strongly recommend you create a MEM. SAV file (see subsection on MEM. SAV later in this section). MEM. SAV allocates a specified number of sectors for storing the RAM resident program, giving you more room to use the DUP. SYS files (see section on DUP. SYS).
 11. Remove your newly created System Diskette from Drive 2 and insert it into Drive 1. Place your Master Diskette into its protective sleeve and store carefully (see section on storing diskettes).
 12. Type **N** and press **RETURN** to create a MEM. SAV file on your System Diskette.
 13. When the prompt message TYPE Y TO CREATE MEM. SAV appears, type **Y** and press **RETURN**.
 14. When the prompt message SELECT ITEM OR RETURN FOR MENU appears, your System Diskette will have a MEM. SAV file on it.

Because DOS II occupies 10K bytes of the available space for data storage, it is not practical to put DOS II on every diskette. Load the diskettes in the following order:

1. Boot with a System Diskette
2. Replace the System Diskette with the program diskette you wish to use.

REMEMBER, if you turn your system off for any reason, you will need to remove any program diskette that does not have DOS II on it and insert the System Diskette before you will be able to reboot the computer.

WRITE- PROTECTING YOUR DISKETTES

Write-protecting is simply a method of preventing you from inadvertently writing over valuable information you may not want to lose from a diskette.

You will notice that the DOS II Master Diskette has no notch on the left side of the diskette jacket so it is impossible to write files on it; therefore, it is already write-protected. Blank and preformatted diskettes do have the notches on the left side of the diskette jacket enabling you to write to the diskette.

Normally, you would not write-protect a System Diskette, as this will defeat the purpose of the MEM. SAV file, preventing you from writing the RAM-resident program to the diskette when necessary. For this reason, you may wish to put the files you want to save on another diskette, which you can write-protect.

HOW TO WRITE-PROTECT VALUABLE DISKETTES

A sheet of large file identification labels and a second sheet of small adhesive write-protect tabs are included in each box of ATARI diskettes.

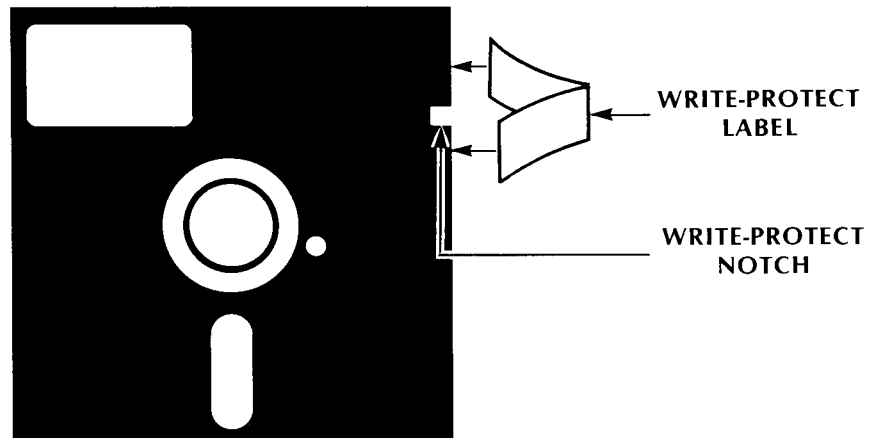


Figure 2-2 Write-Protecting a Diskette

Write-protecting is accomplished by simply removing an adhesive write-protect tab from the sheet and folding it over the notch on the edge of the diskette (see Figure 2-2).

If you are using the ATARI 815 Dual Disk Drive, you may also protect your diskettes by using the write-protect (WRIT PROT) switch on the drive unit. This switch, when on (glows red), prevents you from adding, changing, renaming, or deleting files. When you no longer need the write-protect condition, however, REMEMBER to turn the write-protect switch off.

Should you try writing to a write-protected diskette, you will see an ERROR-144 message displayed on the screen.

LABELING DISKETTES

Use the self-adhesive labels to identify the data on each diskette. Write on the label first to avoid damaging the diskette, and then attach these labels in the upper right corner of the diskette envelope.

SINGLE-DENSITY AND DOUBLE- DENSITY DISKETTE RECORDING

The principle difference between the ATARI 810 Disk Drive and the ATARI 815 Dual Disk Drive is the way the data is encoded for storage on diskettes.

ATARI 810

Information is transferred to the disk drive in blocks of 128 bytes (single density), and each block of 128 bytes fills one sector on the diskette.

ATARI 815

Information is transferred to the disk drive in blocks of 256 bytes (double density), and each block of 256 bytes fills one sector on the diskette.

DOS II is available in two versions:

- 2.0S for recording data on the single-density ATARI 810 Disk Drive
- 2.0D for recording on the double-density ATARI 815 Dual Disk Drive

WHICH DISKETTES TO USE?

To perform disk operations successfully, you must have the correct version of DOS II for your disk drive(s) and the correct blank diskettes for data storage. The chart in Figure 2-3 shows which diskettes you should use depending on the disk drive(s) you have:

ATARI 810 Disk Drive(s)
or
ATARI 815 Dual Disk Drive
or
Combination of ATARI 810 and 815 Disk Drives

DISK DRIVE	CX8104 ATARI 810 MASTER DISKETTE	CX8111 ATARI 810 FORMATTED DISKETTE	CX8100 ATARI 810 BLANK DISKETTE	CX8202 ATARI 810/815 BLANK DISKETTE	CX8201 ATARI 815 MASTER DISKETTE
ATARI 810	X	X	X	X	
ATARI 815				X	X

Figure 2-3 Correct Diskettes for Your Disk Drive

Figure 2-3 shows that you can only use the ATARI 810 Master Diskette II (CX8104) (single-density version of DOS II) with your ATARI 810 Disk Drive. You can choose from ATARI 810 Blank Diskettes (CX8100), ATARI 810/815 Blank Diskettes (CX8202) or ATARI 810 Formatted Diskettes II (CX8111) for program and data storage.

If you have one or more ATARI 815 Dual Disk Drives, use the ATARI 815 Master Diskette (CX8201) (double-density version of DOS II), and the ATARI 810/815 Blank Diskettes (CX8202) that were packed with your disk drive. Since the CX8202 diskettes are totally blank, you will have to format each before using it to duplicate your Master Diskette (see section on formatting).

If you have both ATARI 810 and ATARI 815 Disk Drives attached to your ATARI Personal Computer System, use ATARI 815 Master Diskette (CX8201), and insert it in Drive 1 of the ATARI 815 Dual Disk Drive. MAKE SURE EACH DRIVE has a different drive number setting. Set the ATARI 810 Disk Drive as Drive 3.

HOW TO STORE DISKETTES

Since your diskettes are flexible, they are subject to damage. The following suggestions will help keep your diskettes in good condition:

- ALWAYS keep the diskettes in their protective paper sleeves when not in use.
- Store them vertically as you would properly store records; do not stack them one on top of another.

-
- Store the diskettes AT LEAST 12 inches from your television set or any other possible source of magnetic fields.
 - Store the diskettes away from any direct source of heat.

Your diskettes are an important and valuable part of your ATARI computer system and, with proper care, will give you many hours of dependable use and enjoyment.

USING DOS II

This section and those that follow teach you how to create and work with your files.

IDENTIFYING YOUR DISKETTE FILES

Files are classified into two types:

PROGRAM FILES. These are sets of instructions that tell the computer to perform specific tasks.

DATA FILES. These usually contain the information used by a program file, but not the instructions. For instance, a permanent data file may be a name and address file capable of being updated at any time.

Just as you call a person by name, so must you call a file by a name when you want to access it. The filename on the diskette is part of the file specification (or filespec for short). Filespecs (see Figure 3-1) have six key elements. If you call a file by its wrong name, just like a person, it won't answer; instead, you will see an ERROR-170 appear on the television screen.

The rules for filenames are:

- The maximum length of a filename is eight characters.
- The only characters that can be used in a filename are the letters A through Z, and the numbers 0 through 9.
- The first character in a filename is ALWAYS an alphabetic character.
- The characters * and ? CANNOT be used as part of a name when establishing a filename. (See the section on Wild Cards for explanation.)
- The filenames DOS.SYS, DUP.SYS, AUTORUN.SYS, and MEM.SAV are reserved for DOS II.

FILENAME EXTENDERS AND THEIR USE

You can add a three-character extender to a filename to indicate the type of data in a file. You can use any legal combination of letters and numbers, for example:

SYS	system files
BAS	BASIC program files
DAT	data files
MUS	ATARI Music Composer™ files
ASM	assembly language files
OBJ	binary load files
SRC	source files
LST	files created by the LIST command
SVE	files created by the SAVE command

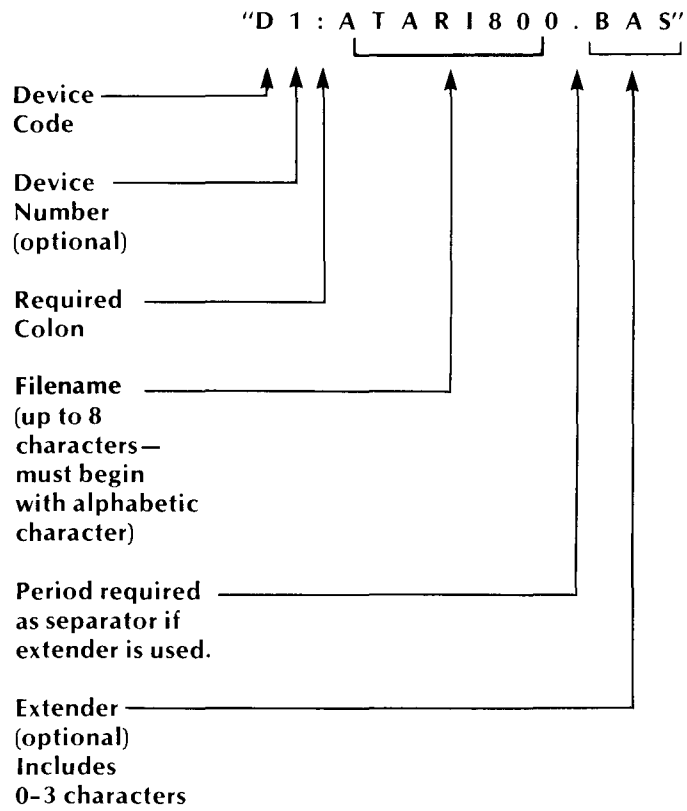


Figure 3-1 Structure of a Filespec

If you try to use an extender that has more than three characters, DOS II will ignore the additional character(s). The example in Figure 3-2 illustrates both legal and illegal filenames, with an explanation of what makes a name illegal.

CASHFLOW	Legal name.
ATARI.BAS	Legal name.
3ATARI.DAT	Illegal name. First character is not an alphabetic letter.
ATARI22.ASM	Legal name.
ATARI#	Illegal name.
A1234567.BA2	Legal name.
B ATARI.LST	Illegal name. No spaces allowed.
DOS.SYS	Illegal name. Reserved for DOS.
DOSSYS	Legal name.
TEST1.123	Legal name.
ATARI.BASIC	Legal name. Note that DOS ignores the last two letters of the extender.

Figure 3-2 Examples of Legal and Illegal Filenames

WILD CARDS

ATARI DOS recognizes two “wild cards” that you can **substitute** for characters in a filename. Wild cards are represented by the special characters, question mark (?) and asterisk (*).

Use the question mark (?) to substitute for a **single** character. The asterisk (*) can stand for any valid combination of characters or number of characters, and therefore, is a great deal more flexible. The following examples illustrate the use of the asterisk and question mark.

Examples:

*.BAS	will list all the program files on a diskette in Drive 1 that end in .BAS.
D2:*. *	will list all the program files on the Drive 2 diskette.
PRO*.BAS	will list all the program files on diskette in Drive 1 that begin with PRO and have .BAS as the extender.
TEST??	will list all the program files on diskette in Drive 1 that begin with TEST and have any combination of letters or numbers for the last two characters.

Figure 3-3 summarizes the DOS Menu options and shows whether they allow you to use wild cards in their parameters.

DOS MENU OPTION	WILD CARDS
A. Disk Directory	Yes
B. Run Cartridge	No
C. Copy File	Yes
D. Delete File	Yes
E. Rename File	Yes
F. Lock File	Yes
G. Unlock File	Yes
H. Write DOS File	No
I. Format Disk	No
J. Duplicate Disk	No
K. Binary Save	No
L. Binary Load	No
M. Run at Address	No
N. Create MEM. SAV	No
O. Duplicate File	Yes

Figure 3-3 DOS Menu Options That Can and Cannot Use Wild Cards

BOOT ERRORS

When you start your system, boot errors can occur for the following reasons (see Figure 3-4):

1. The inserted diskette does not have DOS on it.
2. The diskette was inserted wrong.
3. The diskette has been scratched, warped, or marred. In this case, use another diskette.
4. The diskette is a double-density diskette in an ATARI 810 Disk Drive, or is a single-density diskette in an ATARI 815 Dual Disk Drive.

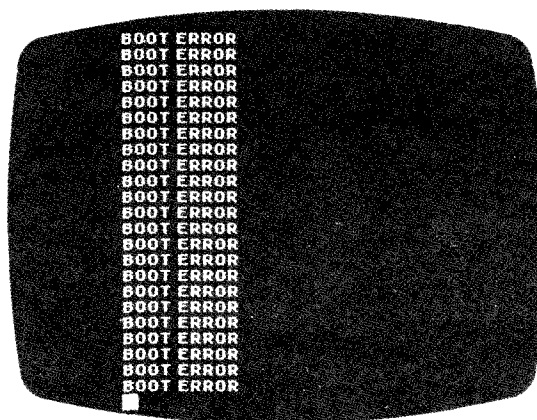


Figure 3-4 Boot Errors

The following conditions will also cause a boot error, but no indication of it will appear on the screen.

1. The disk drive was turned on AFTER the computer console was turned on.
2. The disk drive is not properly connected to the computer console.
3. The power adapter plug has loosened from its wall socket.
4. The power adapter plug has loosened from the disk drive PWR socket.
5. The drive code setting is not correct.

If you have checked, and find none of these problems, take the following steps:

1. Insert the Master Diskette or a System Diskette into Drive 1 and reboot the system.

SAVING, LOADING, AND RUNNING PROGRAMS



2. Remove the Master Diskette and store in a safe place.
3. Reinsert the problem diskette and save any accessible files on another diskette using the process for copying files (see the C. COPY FILE Menu option in Section 4).
4. Then with the problem diskette in Drive 1, use the DELETE FILE(S) function to erase all the files.
5. Try using the diskette again. If this fails, the diskette will have to be reformatted.

Note: For the CX8111 ATARI 810 Formatted Diskette II, this should be done ONLY AS A LAST RESORT to avoid losing the improved formatting.



6. If reformatting fails, the diskette has bad sectors on it and should be discarded.

After you have created your System Diskette and, if necessary, formatted a blank diskette, you are ready to write your own programs. When the power is turned OFF on your computer, you lose any program stored in memory. With an ATARI Disk Drive, you have the means to store and retrieve programs without having to retype them. You can use the following simple BASIC commands to store and retrieve your programs. We have included a sample program for you to type into your computer and a step-by-step procedure for saving it on a diskette and loading it back into the computer. Use the first set of instructions if you have an ATARI 810 Disk Drive and the second set of instructions if you have an ATARI 815 Dual Disk Drive.

If You Have an ATARI 810 Disk Drive:

1. Turn on the disk drive.
2. Insert the System Diskette in Drive 1.
3. Turn on the computer console and television set.
4. When the BUSY light goes off, remove the System Diskette, and insert a diskette that has been formatted.
5. Type the program shown in Figure 3-5.
6. Type **SAVE "D:INTEREST.SAV"** .
7. When the BUSY light goes off and the READY prompt message appears on the screen, the program you typed in is successfully saved on the diskette.
8. Type **NEW**  to erase the program from RAM.

Now to reload the program into memory:

9. Type **LOAD "D:INTEREST.SAV"** .
10. When the READY prompt appears on the screen, you can now run the program by typing **RUN** .

-
11. You can also load and run your program by typing **RUN "D:INTEREST.SAV"**
RETURN.

Note: Do not delete the Sample Interest program from your diskette; you will use it again in Section 5.

```
100 REM *** INTEREST
110 PRINT "IF YOU TYPE THE AMOUNT OF P
RINCIPLE"
120 PRINT "AND THE INTEREST RATE PER Y
EAR, I WILL"
130 PRINT "SHOW YOU HOW YOUR MONEY GRO
WS, YEAR BY"
140 PRINT "YEAR, TO STOP ME, PRESS THE
BREAK KEY."
150 PRINT
160 PRINT "PRINCIPAL";
165 INPUT P
170 PRINT "INTEREST RATE";
175 INPUT R
180 LET N=1
190 PRINT
200 LET A=P*(1+R/100)^N
210 PRINT "YEAR = ";N
220 PRINT "AMOUNT = ";A
230 LET N=N+1
240 GOTO 190

READY
```

Figure 3-5 Sample Interest Program

If You Have an ATARI 815 Dual Disk Drive:

1. Turn on the disk drive.
2. Insert the System Diskette in Drive 1 and a formatted diskette in Drive 2.
3. Turn on the computer and television set.
4. Type the program shown in Figure 3-5.
5. Type **SAVE "D2:INTEREST.SAV"** **RETURN**.
6. When the BUSY light goes off and the READY prompt message appears on the screen, the program you typed is successfully saved on the diskette in Drive 2.
7. Type **NEW** **RETURN** to erase the program from RAM.

Now you are ready to load the program you have saved.

-
8. Type **LOAD "D2:INTEREST.SAV"** **RETURN**.
 9. The program is ready to be run. Type **RUN** **RETURN**.
 10. You can also load and run your program by typing **RUN "D:INTEREST.SAV"** **RETURN**.

Note: Do not delete this program from your diskette; you will use it again in Section 5.

SELECTING A DOS MENU OPTION

To select a DOS Menu option:

1. Type **DOS** and press **RETURN**.
2. The Menu will appear on the screen listing the 15 options available. Refer to Figure 1-4 "The DOS II Menu."
3. Type in your selection and press **RETURN**.
4. A prompt message will appear listing the parameters you need to supply before the DOS can perform the function you have chosen. The parameter is additional information (sometimes optional) specifying how the command is to operate.
5. The prompt message **SELECT ITEM OR RETURN FOR MENU** appears each time the computer system completes a request. If you choose to select another item, type the letter for the option you need and press **RETURN**. The bottom half of the screen will scroll upward to allow the next option's prompt message(s). If you press **RETURN**, the screen will clear and redisplay the DOS Menu.

A. DISK DIRECTORY

The Disk Directory contains a list of all the files on a diskette. On command, it displays the filenames, the extender (if any), and the number of sectors allocated to that file. It will either display a partial list or a complete list depending on the parameters entered. Wild cards can be used in the parameters.

Type **A** and press **RETURN** below the **SELECT ITEM OR RETURN FOR MENU** prompt. The screen immediately displays the entry module message:

DIRECTORY—SEARCH SPEC, LIST FILE

If you press **RETURN** again after this message, you will see a listing of all the filenames on the diskette, the size (in sectors) of each file, and the number of free sectors remaining on the diskette. The following example shows the files in the directory of your system diskette for DOS II:

SELECT ITEM OR RETURN FOR MENU

A **RETURN**

DIRECTORY—SEARCH SPEC, LIST FILE?

RETURN

RETURN

Lists all filenames on screen
from the diskette in Drive 1

DOS SYS 039 (ATARI 810 Disk Drive)

DOS SYS 019 (ATARI 815 Dual Disk Drive)

DUP SYS 042 (ATARI 810 Disk Drive)
DUP SYS 021 (ATARI 815 Dual Disk Drive)

MEM SAV 045 (ATARI 810 Disk Drive)
MEM SAV 022 (ATARI 815 Dual Disk Drive)

581 FREE SECTORS (ATARI 810 Disk Drive)
626 FREE SECTORS (ATARI 815 Dual Disk Drive)

SELECT ITEM OR RETURN FOR MENU

PARAMETERS FOR THE DISK DIRECTORY OPTION

As you can see from the entry module message for the Disk Directory, this command has two parameters:

SEARCH SPEC and LIST FILE

If you do not indicate a specific filespec in this parameter, the DOS will substitute the default values of D1:*.*,E: for these two parameters. The first default parameter, D1:*.*, tells DOS you want to see a listing of all the filenames and file sizes on the diskette currently inserted in Drive 1.

At this time, you can choose to search for a single file, several files, or all files on the diskette you designate. If you do not indicate a specific disk drive, DOS II will assume you want to see the files on the diskette in Drive 1 (the default drive).

The second default parameter, E:, tells DOS you want all this information to be displayed on the screen. Therefore, if you specify neither parameter and simply press **RETURN**, the DOS will list on the screen all filenames and file sizes stored on the diskette inserted in Drive 1.

If you have an ATARI Printer you can print a permanent copy of the directory by using a **P:** for the second parameter. In the example below, the data is printed for only one file, DOS.SYS.

1. Type **A** and press **RETURN**
2. After the directory prompt message, type **DOS.SYS, P:** and press **RETURN**
3. If you have a printer and it is on, a partial directory for Drive 1 will be printed on the printer instead of the screen.

On the screen or hardcopy from the printer you will see:

DOS.SYS 039 (for single density)
DOS.SYS 019 (for double density)

If you do not have a printer (or it is not turned on), you will see an ERROR-138 displayed on the screen. Each time the DOS II DISK DIRECTORY option completes a task, it displays a SELECT ITEM OR RETURN FOR MENU prompt message. Figure 4-1 illustrates several different ways you can use this option.

Note: When filenames are displayed, names and their extenders are separated by a space. However, when you want to access a file, you **MUST** use a period between the filename and its extender.

Example 1:

SELECT ITEM OR RETURN FOR MENU
A **RETURN**
DIRECTORY—SEARCH SPEC, LIST FILE?
*.SYS **RETURN**

Lists all files from Drive 1
diskette with .SYS extender
on the screen.

SELECT ITEM OR RETURN FOR MENU

Example 2:

SELECT ITEM OR RETURN FOR MENU
A **RETURN**
DIRECTORY—SEARCH SPEC, LIST FILE?
D2:,P: **RETURN**

Lists all files on Drive 2
diskette on the line printer.

SELECT ITEM OR RETURN FOR MENU

Example 3:

SELECT ITEM OR RETURN FOR MENU
A **RETURN**
DIRECTORY—SEARCH SPEC, LIST FILE?
EO?.* **RETURN**

Lists all 3-letter filespecs
from the Drive 1 diskette
that begin with EO.

SELECT ITEM OR RETURN FOR MENU

Figure 4-1 Using the Disk Directory Option

B. RUN CARTRIDGE

Whenever you select B, DOS II gives control of your ATARI Personal Computer System to the inserted cartridge. If the BASIC cartridge is inserted, the screen displays a READY prompt. If the Assembler Editor cartridge is inserted, the screen displays an EDIT prompt. If you have not inserted a cartridge, the message NO CARTRIDGE appears on the screen.

Example:

SELECT ITEM OR RETURN FOR MENU
B **RETURN**

If the MEM.SAV file exists on the Drive 1 diskette, your BASIC or assembly language program will automatically be saved to the diskette when you type **DOS RETURN** and then reloaded into RAM when you return control to the cartridge (B. RUN CARTRIDGE). This is assuming the diskette in Drive 1 is the same diskette that was there before you called DOS and that you did not invalidate MEM.SAV by your use of COPY FILE, DUPLICATE FILE, or DUPLICATE DISK. A prompt will appear to remind you that MEM.SAV can be invalidated if you try to use any of these commands (see section on MEM.SAV).

If you did not have a MEM.SAV file on your System Diskette (in Drive 1) when you entered DOS, you will find that any BASIC or assembly language program in memory before you entered DOS is now gone. Your program cannot be recovered now, unless you previously saved it on a diskette before you called DOS. This loss of your program file happens when using DOS II because you share the user program area with the disk utility package stored in the DUP.SYS file. The sharing of RAM with DUP.SYS increases the amount of RAM available to the user compared to DOS I.

C. COPY FILE

Use this option if you have two or more disk drives and want to copy a file from a diskette in one disk drive to another diskette in a second disk drive. There are two parameters associated with the COPY FILE command: FROM and TO. The first parameter, FROM, is usually a filespec, which may or may not contain wild cards. The use of wild cards in the first parameter gives you a very convenient way of copying a group of files from one disk drive to another (see Example 6). The /A option can be used with the second parameter to allow two complementary files to be appended. The second parameter is generally a filespec, but can also be a destination device such as E: (screen), P: (printer), or D: (disk drive) (see Examples 3, 5, and 6 in Figure 4-2).

COPY FILE can also be used to create a backup copy of a particular file on the same diskette with the same filename, but a different extender, or even a completely different filename. If the file you are copying to a new name is made up of several files that have been appended (a "compound" file), the new version of the file will be compressed; i.e., it will take up fewer sectors than the original file from which it was copied.

Note: Attempting to copy any DOS.SYS file will generate an error message. The only way you can write a DOS.SYS file is to use the H. WRITE DOS.SYS file option.

If you attempt to copy a file, as described above, when a MEM.SAV file is on your System Diskette, you will get a new prompt message. You will get the new message after typing in the source drive number (where the information is coming from) and the destination drive number (where the data is going). This message, TYPE Y IF OK TO USE PROGRAM AREA CAUTION: A Y INVALIDATES MEM.SAV, appears to remind you that DOS II can use all of the user program area to speed up the copy file process. A "Y" notifies DOS II that you really don't care about your user program area or MEM.SAV file at this time and MEM.SAV will be invalidated. An **N** response tells the DOS that it cannot put anything into the user program area. It can only use a much smaller, internal buffer to move your file. In other words, your file will still be copied when you give an **N** response, but it will take longer.

You can also use this selection to copy the file listing to the screen (**E:**), or the printer (**P:**)

Caution 1: Do not append tokenized BASIC files, i.e., files stored with a SAVE command. Each tokenized file has its own symbol table, etc., and only the first file will be written. However, you can merge two BASIC files stored with a LIST command, or two binary files created by the Assembler Editor cartridge or DOS II. (Tokenized and untokenized files are explained in Section 5.)

Caution 2: Remember that in merge operations, files stored with a LIST command having matching line numbers could cause the files to interfere with each other.

Example 1:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

D1:DOSEX.BAS, D2:DOSEX.BAS **RETURN**

Copies DOSEX.BAS from D1 to D2.

SELECT ITEM OR RETURN FOR MENU

Example 2:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

D1:DOSEX.BAS,D1:DOSEX.BAK

Creates backup copy of file on same diskette.

SELECT ITEM OR RETURN FOR MENU

Example 3:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

D1:DOSEX.LST,E: **RETURN**

Displays the program listing on screen.

SELECT ITEM OR RETURN FOR MENU

Example 4:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

E:,D1:TEMP.DAT **RETURN**

Copies any succeeding data into a file named TEMP.DAT. Type data on screen that you want to be stored in TEMP.DAT file

PETER **RETURN**

BILL **RETURN**

RAY **RETURN**

STEVE **RETURN**

CTRL 3

Terminates entry of data.

SELECT ITEM OR RETURN FOR MENU

Example 5:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

D1:DISEX.LST,P: **RETURN**

Lists the program listing DISEX.LST on the printer.

SELECT ITEM OR RETURN FOR MENU

Example 6:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

.,D2: **RETURN**

Copies all files from D1 to D2 except those having .SYS extender.

SELECT ITEM OR RETURN FOR MENU

Example 7:

SELECT ITEM OR RETURN FOR MENU

C **RETURN**

COPY—FROM, TO?

D1:PROG2,PROG1/A **RETURN**

Appends PROG2 file on D1 to the PROG1 file.

SELECT ITEM OR RETURN FOR MENU

Figure 4-2 Using the Copy File Option

D. DELETE FILE

This option allows you to delete one or more files from a diskette and the disk directory. Wild cards can be used in the filespec names.

Note: DOS II will not allow you to delete any files on a diskette formatted by DOS I. You must use DOS I to delete files from any DOS I formatted diskette.

The verification prompt message gives you a chance to change your mind about deleting a file. By appending the /N option (No Verification request) to the filespec entry, DOS II will eliminate this verification step (see Example 3 in Figure 4-3).

You can also delete all files on a diskette, but leave the diskette formatted. Example 4 illustrates the steps for deleting all the existing files on the diskette in Drive 1. Note that the /N option is used in this example so the verification request does not need to be answered for each file on the diskette. If you try to delete a locked file, the screen will display ERROR-167 (File Locked).

If you have purchased ATARI 810 Formatted Diskettes II, this is an excellent way to clear the diskette of all files without destroying the improved formatting available on these diskettes.

E. RENAME FILE

This option allows you to change the name of one or more files. There are two parameters, OLD NAME and NEW, for this option. The parameter OLD NAME is always a complete filespec. If you do not specify a device number, the computer assumes D1: (the default). The NEW parameter refers simply to the new filename. The device number is automatically the device specified in the OLD NAME parameter. If there are any illegal characters in the parameter NEW, the name of the renamed file will consist of the characters up to, but not including, the illegal character. You can use wild cards in both the first and second parameters (see Example 2 in Figure 4-4).

Warning: Do NOT rename any file on a DOS II diskette using DOS I. In general, you should never use DOS I with DOS II diskettes.

If you attempt to rename a file on a write-protected diskette, an ERROR-144 (Device Done Error) will display on the screen. If you try to rename a file that is not on the diskette, an ERROR-170 (File Not Found) error displays. If the screen displays ERROR-167, it means that you tried to rename a locked file (see F. LOCK FILE).

Example 1:

SELECT ITEM OR RETURN FOR MENU

D

DELETE FILESPEC

D2:REM*.BAS

TYPE "Y" TO DELETE...

REM1.BAS?

Y

REMBAA. BAS

Y

SELECT ITEM OR RETURN FOR MENU

All files that begin with REM and that have a .BAS extender. Verification prompt. Deletes REM1.BAS.

Deletes REMBAA.BAS.

Example 2:

SELECT ITEM OR RETURN FOR MENU

D

DELETE FILESPEC

D: TEMP.DAT

TYPE "Y" TO DELETE...

TEMP.DAT

N

SELECT ITEM OR RETURN FOR MENU

A single file. Verification prompt.

If Y is typed, file will be deleted.

Example 3:

SELECT ITEM OR RETURN FOR MENU

D

DELETE FILESPEC

DOXEX.BAS/N

SELECT ITEM OR RETURN FOR MENU

File will be deleted without requesting verification.

Example 4:

SELECT ITEM OR RETURN FOR MENU

D

DELETE FILESPEC

. /N

SELECT ITEM OR RETURN FOR MENU

Deletes all files from the Drive 1 diskette.

Figure 4-3 Using the Delete File Option

Example 1:

SELECT ITEM OR RETURN FOR MENU

E

RENAME, GIVE OLD NAME, NEW

D2: TEMP.DAT,NAMES.DAT

Changes the file on Drive 2
from TEMP.DAT to
NAMES.DAT.

SELECT ITEM OR RETURN FOR MENU

Example 2:

SELECT ITEM OR RETURN FOR MENU

E

RENAME, GIVE OLD NAME, NEW

.8KB,.BAS

All files with extender 8KB have
their extenders changed to .BAS

SELECT ITEM OR RETURN FOR MENU

Figure 4-4 Using the Rename File Option

Example 1:

SELECT ITEM OR RETURN FOR MENU

F

WHAT FILE TO LOCK?

DOS.SYS

Locks the DOS.SYS File.

SELECT ITEM OR RETURN FOR MENU

Example 2:

SELECT ITEM OR RETURN FOR MENU

F

WHAT FILE TO LOCK?

D1:*.BAS

Locks all files on D1 with an
extender of .BAS.

SELECT ITEM OR RETURN FOR MENU

Example 3:

SELECT ITEM OR RETURN FOR MENU

F

WHAT FILE TO LOCK?

T*.*

Locks all files on D1 that
begin with T.

SELECT ITEM OR RETURN FOR MENU

Example 4:

SELECT ITEM OR RETURN FOR MENU

F

WHAT FILE TO LOCK?

.

Locks all D1 files.

SELECT ITEM OR RETURN FOR MENU

Figure 4-5 Using the Lock File Option

F. LOCK FILE

Use this selection to write-protect a single file. A locked file cannot be written to, appended, renamed, or deleted. An ERROR-167 will result from trying to write to a locked file. You can use wild cards to lock several files at the same time.

A locked file will appear on the Disk Directory with an asterisk (*) preceding its name. DO NOT confuse this asterisk with a wild card.

Warning: If you lock any files on the Disk Directory and then format the diskette, the locked files WILL STILL BE OBLITERATED. In other words, formatting ignores the LOCK FILE command.

G. UNLOCK FILE

Use this option to unlock a file or files you previously locked using option F. When you complete this option, the asterisk that appeared before the filename in the Disk Directory to indicate the file was locked, will no longer appear on the screen the next time you execute a DISK DIRECTORY command (DOS Menu option A.). Wild cards can be used in the filespec names.

Example 1:

SELECT ITEM OR RETURN FOR MENU

G

WHAT FILE TO UNLOCK?

DOSEX.BAS

Unlocks DOSEX.BAS file on D1.

SELECT ITEM OR RETURN FOR MENU

Example 2:

SELECT ITEM OR RETURN FOR MENU

G

WHAT FILE TO UNLOCK?

T*.*

Unlocks files beginning with the letter T on Drive 1.

SELECT ITEM OR RETURN FOR MENU

Example 3:

SELECT ITEM OR RETURN FOR MENU

G

WHAT FILE TO UNLOCK?

PROB?.DAT

Unlocks all 5-letter files beginning with PROB and having a .DAT extender.

SELECT ITEM OR RETURN FOR MENU

Figure 4-6 Using the Unlock File Option

H. WRITE DOS FILE

To write DOS II (composed of DOS.SYS and DUP.SYS files) onto a diskette, you must have previously formatted the diskette using DOS II (see I. FORMAT DISK) or else be using an ATARI 810 Formatted Diskette II (CX8111). (The diskette on which DOS II is to be written can be inserted in the disk drive of your choice.)

Note: DOS II will not allow you to write the new DOS II files onto a diskette formatted by DOS I. Similarly, DOS I should only be used to WRITE DOS FILE onto a DOS I formatted diskette. Even though we advise against it, you should be aware

that DOS I does allow you to write a copy of DOS I onto a DOS II formatted diskette.

Warning: You should never WRITE DOS FILE onto a DOS II diskette. When working with some diskettes formatted by both DOS I and some formatted with DOS II, use DOS II to protect yourself from making errors that might damage your valuable diskettes.

As soon as the DOS files have been written to the diskette (see Figure 4-7), the screen is cleared and both the menu and prompt message, SELECT ITEM OR RETURN FOR MENU, are redisplayed.

If you try to write a new DOS file onto a diskette that has been write-protected, you will get an ERROR-144. You will also get an error if you try writing a new DOS file onto a diskette inserted in an 815 Disk Drive that has the WRIT PROT light on.

```
SELECT ITEM OR RETURN FOR MENU
H 
DRIVE TO WRITE DOS FILES TO?
1 
TYPE "Y" TO WRITE DOS TO DRIVE 1.
Y 
WRITING NEW DOS FILES

SELECT ITEM OR RETURN FOR MENU
```

Figure 4-7 Using the Write DOS File Option

I. FORMAT DISKETTE

This option is used to format a diskette. The diskette can be blank or have files on it that you no longer want. Formatting writes a digital pattern on the diskette that allows data to be stored and retrieved. Formatting a diskette takes approximately 2 minutes on the ATARI 810 Disk Drive and 2 1/2 minutes on the ATARI 815 Dual Disk Drive.

The example in Figure 4-8 illustrates Drive 1 as the drive to be formatted; however, you can specify any drive. It is not possible to format a diskette containing bad sectors. The screen will display an ERROR-173 (Bad Sectors at Format Time), and DOS II will refuse to format the diskette. If DOS II gets a message from the disk drive that the diskette has bad sectors, it will try to format the diskette two additional times. If this happens, it may take up to 15 minutes trying to format a diskette before returning an ERROR-173.

If a diskette is new and has bad sectors, it is recommended you return it to the supplier for exchange. You should be aware that diskettes supplied by vendors other than ATARI may not be of high enough quality to work with the ATARI Disk Drives.

```
SELECT ITEM OR RETURN FOR MENU
I 
WHICH DRIVE TO FORMAT?
1 
TYPE "Y" TO FORMAT DISK 1
Y 
SELECT ITEM OR RETURN FOR MENU
```

Figure 4-8 Using the Format Disk Option

Warning: Formatting a diskette always destroys all files and format existing on the diskette. If you format an ATARI 810 Formatted Diskette II (CX8111), you will lose the speed advantage of using a preformatted diskette. Use Menu Option D. DELETE FILE(S) instead.

J. DUPLICATE DISK

Use this menu option to create an exact duplicate of any diskette. You can use this option with a single disk drive by manually swapping source (diskette with files on it) and destination (diskette on which you are putting files) until the duplication process is complete. You can also use this option with multiple disk drive systems by inserting source and destination diskettes in two separate drives and allowing the duplication process to proceed automatically.

The duplication process is a sector-by-sector copying technique. This means that not only are all your files copied from the source to the destination diskette, but they are also located in the same sector number on both diskettes. The directory of the source diskette is also copied onto the destination diskette. For this reason, any files previously stored on the destination diskette will have been destroyed when the duplication process is complete.

The source diskette must be a DOS II formatted diskette for you to use this option. If you attempt to use a DOS I diskette, you will get an error message. The destination diskette can be any formatted ATARI diskette. This means you can use any ATARI 810 Formatted Diskette II (CX8111) or any diskette formatted on your disk drive using DOS I or DOS II. If you use an old diskette for the destination diskette, however, be sure none of the files on it are valuable, because the duplication process will write over them with the new files.

Remember, you cannot duplicate a DOS I formatted diskette, nor can you duplicate diskettes between an ATARI 810 Disk Drive and an ATARI 815 Disk Drive. If you try to do so, you will get an error message.

You can still approximate the duplication process from an ATARI 815 Dual Disk Drive to an ATARI 810 Disk Drive (and vice versa) and from DOS I to DOS II diskettes by using the COPY FILE command with the *.* option discussed earlier. The data will look the same on each diskette, but there are differences in the way data is stored.

Since there is no true duplication process between DOS I and DOS II diskettes, or between ATARI 810 Disk Drives and ATARI 815 Dual Disk Drives, you should always save BASIC or assembly language programs that are currently in RAM before attempting to duplicate a diskette. There is no internal buffer for DUPLICATE DISK as there is for the COPY FILE command, and MEM.SAV will be invalidated if you give DOS II permission to proceed (and to use the program area). The DUPLICATE DISK option always uses the program area (where a RAM-resident BASIC program is stored) as a buffer for moving the files on the source diskette to the destination diskette when one drive is used.

DUPLICATION USING A SINGLE DISK DRIVE

In a single disk drive system, the source and destination drives are both Drive 1 (see Figure 4-9).

Always write-protect your source diskette as a safety measure. Then, if it is accidentally inserted in place of the destination diskette, the screen will display an ERROR-144, and your source diskette will still be intact.

If you type any character other than Y in response to the TYPE "Y" IF OK TO USE PROGRAM AREA message, the program aborts and the SELECT ITEM OR RETURN FOR MENU prompt appears on the screen.

Figure 4-9 is an example of duplication using a single disk drive:

```
SELECT ITEM OR RETURN FOR MENU
J 
DUP DISK—SOURCE, DEST DRIVES?
1,1 
INSERT SOURCE DISK, TYPE RETURN
TYPE "Y" IF OK TO USE PROGRAM AREA?
CAUTION: A "Y" INVALIDATES MEM. SAV
Y 
INSERT DESTINATION DISK, TYPE RETURN

SELECT ITEM OR RETURN FOR MENU
```

Figure 4-9 Using the Duplicate Disk Option With a Single Disk Drive

Note: The number of times the DUP program requests you to insert the source and destination diskettes depends on the number and size of the file(s) to be duplicated for a given system and the amount of RAM in the system. To copy a diskette that is full, a 48K system might require only two insertions, whereas a 16K system might require five or six diskette insertions.

DUPLICATION USING MULTIPLE DISK DRIVES

If you are using both the ATARI 810 and ATARI 815 Disk Drives, make sure you distinguish between files stored using the single-density and double-density formats when labeling the diskettes. This will keep you from using them in the wrong disk drive.

For a multiple disk drive system, it is also necessary to save a RAM-resident BASIC program, as the user's program area will be altered and MEM.SAV will be invalidated. Notice that the source diskette is inserted in Drive 1 and the destination diskette into Drive 2 (Figure 4-10). You can use any two of the same disk drive models.

The cursor remains on the screen during the duplication process. This process can take several minutes if the source diskette is almost full.

```
SELECT ITEM OR RETURN FOR MENU
J 
DUP DISK—SOURCE, DEST DRIVES
1,2 
INSERT BOTH DISKS, TYPE RETURN

TYPE "Y" IF OK TO USE PROGRAM AREA
CAUTION: A "Y" INVALIDATES MEM. SAV
Y 
SELECT ITEM OR RETURN FOR MENU
```

Figure 4-10 Using the Duplicate Disk Option With Dual or Multiple Disk Drives

K. BINARY SAVE

Note: This instruction will probably not be used by a beginning ATARI Personal Computer user. Unless you understand hexadecimal numbers and have some knowledge of assembly language, you may not wish to read the information beyond the first example.

Use this Menu selection to save the contents of memory locations in object file (binary) format. Programs written using the Assembler Editor cartridge also have this format. The parameters for this selection: START, END, INIT, RUN, are hexadecimal numbers. The START and END addresses are required parameters for any binary file or program. The INIT (initialize) and RUN addresses are optional parameters that allow you to make any program execute on loading. See Examples 2, 3, and 4.

In the example below, a file to be called BINFIL.OBJ with the starting address 3C00 and the ending address 5BFF is saved on a diskette in Drive 1.

Figure 4-11 is an example of the use of BINARY SAVE:

Example 1:
SELECT ITEM OR RETURN FOR MENU
K **RETURN**
SAVE—GIVE FILE, START, END, INIT, RUN
BINFIL.OBJ, 3C00, 5BFF **RETURN**
SELECT ITEM OR RETURN FOR MENU

Figure 4-11 The Most Elementary Use of Binary Save

ADVANCED
USER
INFORMATION
ABOUT
OPTIONAL
PARAMETERS

All binary files, like those you would create with the BINARY SAVE option or with the Assembler Editor cartridge, have a common 6-byte header that precedes the file (see Figure 4-12). From the header data shown in the table, you can easily pick out the starting address and ending address that was used in Figure 4-12.

Header Byte #	Decimal Number	Hex Number	Description
#1	255	FF	Identification code for binary load file
#2	255	FF	
#3	0	00	Starting address (LSB) (MSB)
#4	60	3C	
#5	255	FF	Ending address (LSB) (MSB)
#6	91	5B	
.	.	.	File data segment contains 8191 (Dec) bytes of data.
.	.	.	

Figure 4-12 Six-Byte Header Table for Binary Save

The two optional parameters, INIT and RUN, offer the means to make a binary assembly language file execute automatically after loading. A file that makes use of either or both of these address parameters is called a “load-and-go” file. A file that does not contain data for these parameters is called a “load” file, since it loads into the computer but will not execute until a M. RUN AT ADDRESS command is given.

In general, the RUN address parameter defines the point in a program where execution will begin as soon as a whole file is loaded into RAM (i.e., when End of File is reached). For this reason there can only be one effective RUN address even if a file is a compound file. For example, a file could be made up of several small files appended together with each of the original small files having their own RUN address. In this case, only the last RUN address to be loaded would execute.**

If an INIT address is specified, then as soon as the actual address gets loaded into RAM, the code that it points to will be executed. This is true even if the file is made up of several load-and-go files appended together. In such a case each load-and-go segment which has an INIT address specified will be executed when the INIT address is loaded. Thus, each segment would load and be executed before the next segment would be loaded, etc.* Execution of code pointed to by any INIT address always precedes the execution of any code pointed to by a RUN address.

Files created by the Assembler Editor cartridge using the load-and-go option can be stored in the desired INIT and RUN addresses in your code followed by the code to be controlled. The RUN address is always stored in Locations 2E0 (LOW) and 2E1 (HIGH) Hex. The INIT address is always stored in Locations 2E2 (LOW) and 2E3 (HIGH) Hex. Remember, the INIT address is executed as soon as it is loaded, so the code that it points to must have been previously loaded.

Note: IOCB #1 is open during the execution of code pointed to by any INIT address. For this reason it is not available and must not be tampered with by the user program being executed.

**An RTS (RETURN) at the end of a program will always return control to DOS II.

*Each code segment must end with an RTS (RETURN) if the next segment is to be loaded or, if it is desired, returned to DOS II control.

USING BINARY SAVE WITH OPTIONAL PARAMETERS

The example in Figure 4-13 illustrates an assembly language program that uses a data area that must be initialized before the main program can use it. Suppose the initialization code resides from address 4000 (Hex) to 41FF (Hex) and the main program resides between 4200 (Hex) and 4FFF (Hex). For purposes of illustration, assume that both the initialization code and main program contain executable code and the initialization code ends with an RTS (RETURN).

In the following example we assume the program, LAGPRG.OBJ, is already in memory.

Example 2:

SELECT ITEM OR RETURN FOR MENU

K **RETURN**

SAVE—GIVE FILE, START, END, INIT, RUN

LAGPRG.OBJ, 4000, 4FFF, 4000, 4200 **RETURN**

SELECT ITEM OR RETURN FOR MENU

Figure 4-13 Using Binary Save With Optional Parameters

The following events will occur on loading this file into memory:

1. Memory from 4000 to 4FFF will be filled with the program.
2. The INIT address 4000 (Hex) is stored in Memory Locations 2E2 and 2E3 (Hex).

3. Initialization program from 4000 to 41FF will execute.
4. The RUN address 4200 (Hex) is stored in Memory Locations 2E0 and 2E1 (Hex).
5. Main program from 4200 to 4FFF begins to execute and will continue to do so until a RETURN (RTS) is executed, or a **SYSTEM RESET** or **BREAK** occurs.

In the case of compound files, the result is more complicated, depending on how the now appended files were created. The next section illustrates several cases where files have been appended.

STRUCTURE OF A COMPOUND FILE

Before considering the next example, look at the structure of a compound file. A compound file is constructed of various binary files that have been appended together. This can be done in one of two ways. One way is to use the C. COPY FILE option with its append option. A compound file created with this command is not compatible with the Assembler Editor loader, although it can be loaded using the L. BINARY LOAD option of DOS II. If compatibility with the Assembler Editor cartridge is desired, an alternate way to create a compound file is to use the K. BINARY SAVE option we have been discussing. The two types of files are illustrated in Appendix I. The only real difference is that the FFFF (Hex) identification code is included with every segment when a compound file is created using C. COPY File.

When K. BINARY SAVE is used, the additional identification codes for each segment (after the first one) are NOT included in the final file. This is the only form of compound file that is compatible with the Assembler Editor cartridge. The L. BINARY LOAD option of DOS II, however, is compatible with both types of compound files.

Now consider what happens when a compound file like this is loaded—supposing various INIT and RUN addresses were specified for each of these files before they were appended. (It will help you to think of the INIT and RUN addresses as being part of the data in each segment, which they are essentially.)

Example 3:

Suppose you have three files, each of which has a RUN address, but no INIT address included in its data. The example in Figure 4-14 shows one way a file of this type might be created.

```
SELECT ITEM OR RETURN FOR MENU
K RETURN
SAVE FILE—GIVE FILE, START, END, INIT, RUN
PART1.OBJ, 2000, 21FF,, 2000 RETURN
SELECT ITEM OR RETURN FOR MENU

SELECT ITEM
K RETURN
SAVE ITEM—OR RETURN—FOR MENU
PART2.OBJ/A, 2200, 23FF,, 2200 RETURN
SELECT ITEM OR RETURN FOR MENU
```

Figure 4-14 Using Binary Save to Save Compound Files

The other two files, PART2.OBJ and PART3.OBJ that are created the same way as PART1.OBJ, can then be merged into WHOLE.OBJ by using the K. BINARY SAVE or C. COPY FILE option with the append option. What happens now when this new file is loaded?

1. PART1.OBJ loads, but does not execute (no INIT).
2. RUN address for PART1.OBJ is stored in 2E0 and 2E1.
3. PART2.OBJ loads, but does not execute (no INIT).
4. RUN address for PART2.OBJ is stored in 2E0 and 2E1, which overwrites PART1.OBJ RUN address.
5. PART3.OBJ loads, but does not execute (no INIT).
6. RUN address for PART3.OBJ is stored in 2E0 and 2E1, which overwrites PART2.OBJ RUN address.
7. Execution begins at RUN address of PART3.OBJ since you are now at the end of the file.

Example 4:

For another example of a compound file (Figure 4-15), consider a three-segment file, BIGFILE.OBJ. Suppose each segment loads into a different area of memory and that

SEG1.OBJ has an INIT address, but no RUN address,

SEG2.OBJ has no INIT or RUN address,

SEG3.OBJ has an INIT address and a RUN address for SEG2.OBJ and, in addition, is loaded on top of SEG1.OBJ.

When BIGFILE.OBJ is loaded, the following events occur.

SEG1.OBJ is loaded.

SEG1.OBJ executes starting at its INIT address.

SEG2.OBJ is loaded.

SEG3.OBJ is loaded on top of SEG1.OBJ.

SEG3.OBJ executes starting at its INIT address.

SEG2.OBJ executes starting at the RUN address specified in SEG3.OBJ.

Clearly, this option gives you great power and flexibility for creating large files that load and execute immediately.

Example 5:

To convert an existing load-only file to a load-and-go file, you can load the file into memory and then save it under a new filename using the K. BINARY SAVE Menu option. This poses some problems, as you can sometimes forget the final address the file occupies, or the file could be compounded with the segments not necessarily consecutive in memory. Therefore, the new file would take up more space on the diskette than the old, etc. You can avoid these problems by using the procedure shown in the following example. This example illustrates a load file with a run address of 4000 Hex that is changed to a load-and-go file.

In Figure 4-15, a one-byte file located at FF00 (in the O. S. ROM) is appended to the end of your file LOADFIL.OBJ. Since this file's run address is the same as the address at which your load file normally runs, your load file begins execution as soon as the entire appended file is loaded into RAM.


```
SELECT ITEM OR RETURN FOR MENU
K 
SAVE FILE — GIVE FILE, START, END, INIT, RUN
LOADFIL. OBJ/A, FF00, FF00,, 4000
SELECT ITEM OR RETURN FOR MENU
```

Figure 4-15 Converting an Existing Load-Only File to a Load-and-Go File



L. BINARY LOAD

Note: This instruction will probably not be used by a beginning ATARI Personal Computer user.



Use this selection to load into RAM an assembly language (binary) file that was previously saved with menu option K. or created by the Assembler Editor cartridge. If the RUN address or INIT address was appended to the file in Locations 2E0 and 2E1 or 2E2 and 2E3, the file will automatically run after being entered. In a load-and-go file, INIT and RUN addresses are ignored when you type /N after the filename (see Example 1 in Figure 4-16). The file can then be run using the RUN AT ADDRESS Menu option.

An example of using this option without the /N option is shown in the second example in Figure 4-16. Since this file had the starting address in Locations 2E0 and 2E1 appended to it (see Example 1 for K. BINARY SAVE), the file will begin executing as soon as the load is complete.

Example 1:

```
SELECT ITEM OR RETURN FOR MENU
L 
LOAD FROM WHAT FILE?
MYFILE. OBJ/N 
SELECT ITEM OR RETURN FOR MENU
```

Example 2:

```
SELECT ITEM OR RETURN FOR MENU
L 
LOAD FROM WHAT FILE?
BINFIL. OBJ 
```

Example 3:



```
SELECT ITEM OR RETURN FOR MENU
L 
LOAD FROM WHAT FILE?
MACHL. OBJ 
SELECT ITEM OR RETURN FOR MENU
```

Figure 4-16 Using the Binary Load Option

Example 3 in Figure 4-16 illustrates a file called MACHL.OBJ that does not have a RUN address or an INIT address. In this case, the SELECT ITEM OR RETURN FOR MENU prompt message will display on the screen as soon as the file finishes loading.

To execute a file that has no appended RUN or INIT address, see the next menu option, M. RUN AT ADDRESS.

M. RUN AT ADDRESS

Note: This instruction will probably not be used by a beginning ATARI Personal Computer user.

Use this selection to enter the hexadecimal starting address of an object file program after you have loaded it into RAM with the BINARY LOAD selection. This selection is used when the starting address has not been appended to the object file.

In Figure 4-17, the instructions at hexadecimal Location 3000 will begin executing. Be very careful when entering these hexadecimal address locations. If you enter an address that does not contain executable code, it will create problems. As an example, you could lock up the system, making it necessary for you to reboot.

SELECT ITEM OR RETURN FOR MENU

M **RETURN**

RUN FROM WHAT ADDRESS?

3000 **RETURN**

Figure 4-17 Using the Run at Address Option

N. CREATE MEM.SAV

This option allows you to create a file on diskette called MEM.SAV into which the contents of lower user memory are saved whenever you call DOS. When you type **DOS** **RETURN**, the computer saves the RAM-resident user program (if any) in the MEM.SAV file before it brings the diskette file DUP.SYS into RAM. When you have finished using the DOS options, you simply return control to the cartridge by typing **B** **RETURN**, and MEM.SAV will automatically reload your program into RAM. If you are not using a cartridge, typing **B** has no effect. You will have to respond to the SELECT ITEM OR RETURN FOR MENU prompt.

You must be careful not to allow DOS to use all of user memory when you want the COPY FILE, DUPLICATE FILE, or DUPLICATE DISK options for saving the existing data. DOS does not know if ALL or only part of your program has been saved in MEM.SAV. When DOS utilizes all of user memory, it automatically invalidates the MEM.SAV file. If this occurs, your program will not be reloaded when control is returned to the cartridge.

SELECT ITEM OR RETURN FOR MENU

N **RETURN**

TYPE "Y" TO CREATE MEM.SAV

Y **RETURN**

SELECT ITEM OR RETURN FOR MENU

Figure 4-18 Creating a MEM.SAV File

If you attempt to use this option to create a MEM.SAV file on a diskette that already has a MEM.SAV file, the screen will display the message MEM.SAV FILE ALREADY EXISTS and follow it with the prompt message SELECT ITEM OR RETURN FOR MENU. Figure 4-18 illustrates the steps for creating a MEM.SAV file on a diskette inserted in Drive 1. Note that MEM.SAV files can only be created on a diskette in Drive 1.

WHY HAVE A MEM.SAV FILE?

This special file allows you to save your RAM-resident program temporarily in a special file on diskette. To be effective, MEM.SAV (which requires 45 sectors) must be on the diskette inserted in Drive 1. This diskette must not be write-protected if MEM.SAV is to work. Once MEM.SAV exists on your diskette, then the area of user memory to be overwritten by DUP.SYS will be stored in MEM.SAV every time DOS is called. Essentially, you are performing a swap contents operation, thereby “expanding” your user program area. This swap takes about 21 seconds. When you return control of the computer system to the cartridge, the DUP.SYS file is in turn overwritten as the contents of MEM.SAV are loaded back into RAM automatically. This operation takes about seven seconds.

If you are working on a BASIC program and need to return to DOS for some reason, you can do so using MEM.SAV without having to save your program to diskette and reenter it. When you finish using DOS and return control of the computer system to the cartridge, the MEM.SAV file is automatically reloaded into memory and your BASIC program is restored into user program memory.

An example of MEM.SAV usage is in Figure 4-19.






1. Type **LOAD “D:MYPROG. BAS”** .
2. Edit your program and then type **RUN** .
3. It works and you want to **RENAME** the original file to keep as a backup copy.
4. Type **DOS** .
5. Make your Menu selection (**E** for **RENAME FILE**) and rename your original file to MYPROG. OLD.
6. Type **B**  to return to BASIC. With the help of MEM.SAV, your modified version of MYPROG.BAS is automatically reloaded into RAM.
7. Type **SAVE “D:MYPROG.BAS”**  to save your modified program under the original name.

Figure 4-19 Example of MEM.SAV Usage

USING MEM.SAV TO WRITE ASSEMBLY LANGUAGE PROGRAMS

The MEM.SAV file also allows you to write assembly language programs (or load in binary data) that share the user program area with DUP.SYS. This means you are free to write programs or load data in the area from LOMEM (which fluctuates with the number of drives in the system and the number of files that can be open concurrently) to HIMEM (which fluctuates depending on which Graphics Mode you are in). See Appendix C, Memory Map.

Example:

Suppose you have a binary file you want to execute automatically as soon as it is loaded. This type of file is called a load-and-go file. The run address is already pro-

grammed into such a file and you will not need to select the RUN AT ADDRESS option. In this case, it is not necessary to have a MEM.SAV file on your diskette. Since the file is load-and-go, it will simply load and then begin to execute. The safest way to get back to DOS is to reboot your computer. If you have not overwritten the DUP.SYS program during the execution of your binary file, then you can recover by simply executing a RETURN (RTS) in your program. If your binary file overwrites DUP.SYS during the time it is loading, DOS will keep track of this fact and will automatically reload and execute DUP.SYS after the RETURN in your program is executed.

Warning: If the execution of your load-and-go file writes into any areas below LOW MEM used by DOS.SYS, DUP.SYS, or a RAM area used by the Operating System, the RETURN (RTS) from your program may leave the computer in an undefined state. Should this occur, you may have to power up the computer again to recover.

USING MEM.SAV TO LOAD BINARY FILES

This section deals with loading a binary file that is not to be executed at the same time it is loaded, or loading a file that contains data for another program. If your LOAD file does not overlay any part of the DUP.SYS area, then a MEM.SAV file is not required.

If your LOAD file overlays any part of the DUP.SYS file, you must have a MEM.SAV file on the diskette in Drive 1 if the load is to be successful. If you do have MEM.SAV, the following actions take place after you execute an L. LOAD BINARY FILE option:

1. You use the LOAD BINARY FILE selection to load your file.
2. Your original MEM.SAV is loaded from disk into memory overlaying and invalidating DUP.SYS.
3. Your file is loaded on top of the original MEM.SAV modifying part or all of the original MEM.SAV file.
4. Your new MEM.SAV file in RAM is saved in the MEM.SAV area on the diskette.
5. DUP.SYS is reloaded from diskette into memory.
6. You remain in DOS until you choose either to:

RUN CARTRIDGE	at which time your file is loaded into memory from MEM.SAV and you come up under the control of your BASIC or assembly language cartridge.
---------------	--

RUN AT ADDRESS	at which time your file is loaded into memory from MEM.SAV and you begin execution of whatever code is at the address you specified.
----------------	--

LOAD BINARY FILE	where you wish to load a load-and-go file. In this instance, if the new file also overlays a part of DUP.SYS, but not the original file, then both MEM.SAV and your new file will now be in memory when the load is complete. If the new file does not overlay DUP.SYS at all,
------------------	--

then the load will complete with only the new file loaded into RAM. Since the new file is a load-and-go and loaded whether DUP. SYS is overlaid or not, you will come up under the control of this file until a RETURN (RTS) is executed.

Note: If you wish to have two files in memory simultaneously, one of which resides wholly or in part in the DUP.SYS area and the other of which resides wholly outside of the DUP.SYS area, the easiest way to achieve this is by merging the two files into one file and then loading the newly merged file.

O. DUPLICATE FILE

This option is used if you have only one disk drive and want to copy a file from one diskette to another. Remember that a single disk drive must always be set up as Drive 1. Since there is only one disk drive, you must manually insert and remove the source and destination diskettes. If a file is very long, you may have to alternate the source and destination diskettes several times before the duplication process is complete. Allowing DOS to take over user memory will reduce the amount of diskette switching required to copy long files. However, you must remember this will mean that your MEM.SAV file will be invalidated.

Wild cards are available with this option. In Example 2 you will notice that even if you do give DOS permission to take over user memory (with a wild card filename), your files are still copied one at a time. You will have to alternate diskettes at least once for each filename that you want to copy.

The second example illustrates using a wild card to copy files having five letters and beginning with TEST from one diskette to another. This example assumes the source diskette has only two files with names that satisfy TEST?

In Example 3, both the filename and extenders have been replaced with wild cards. DOS will therefore copy all files except those that have an extender of .SYS. This example assumes only three files are to be copied: MEM.SAV, TEST1, and TEST2.





Example 1:


```
SELECT ITEM OR RETURN FOR MENU
O 
NAME OF FILE TO MOVE?
DOSEX. BAS 
TYPE "Y" IF OK TO USE PROGRAM AREA
CAUTION: A "Y" INVALIDATES MEM.SAV
Y 
INSERT SOURCE DISK, TYPE RETURN


INSERT DESTINATION DISK, TYPE RETURN



SELECT ITEM OR RETURN FOR MENU
```

Example 2:





```
SELECT ITEM OR RETURN FOR MENU
O 
NAME OF FILE TO MOVE?
TEST? 
TYPE "Y" IF OK TO USE PROGRAM AREA
CAUTION: A "Y" INVALIDATES MEM.SAV
Y 
INSERT SOURCE DISK, TYPE RETURN




    COPYING---D1: TEST1
INSERT DESTINATION DISK, TYPE RETURN




INSERT SOURCE DISK, TYPE RETURN


    COPYING---D1: TEST2
INSERT DESTINATION DISK, TYPE RETURN

INSERT SOURCE DISK, TYPE RETURN

SELECT ITEM OR RETURN FOR MENU
```

Example 3:

```
SELECT ITEM OR RETURN FOR MENU
O 
NAME OF FILE TO MOVE?
* * 
TYPE "Y" IF OK TO USE PROGRAM AREA
CAUTION: A "Y" INVALIDATES MEM.SAV
Y 
INSERT SOURCE DISK, TYPE RETURN


    COPYING—D1: MEM.SAV
INSERT DESTINATION DISK, TYPE RETURN

INSERT SOURCE DISK, TYPE RETURN


    COPYING---D1: TEST1
INSERT DESTINATION DISK, TYPE RETURN

INSERT SOURCE DISK, TYPE RETURN




    COPYING---D1: TEST2
INSERT DESTINATION DISK, TYPE RETURN

INSERT SOURCE DISK, TYPE RETURN

SELECT ITEM OR RETURN FOR MENU
```

Figure 4-20 Using the Duplicate File Option

MORE USER INFORMATION

BASIC COMMANDS USED WITH DOS

Before describing the BASIC commands used with DOS II, you need to know how the commands will act on programs being stored and retrieved. The following paragraphs explain the two types of files that can hold BASIC programs.

TOKENIZED AND UNTOKENIZED FILES

The first type of file, called "untokenized," contains standard ATASCII text characters so it looks like a printout of a BASIC program. These programs do not retain their symbol tables each time they are loaded and saved. The symbol table associates the variable name with the memory location where the values for that variable are stored. To store and retrieve a file in its untokenized form, you use the LIST and ENTER commands.

The second type, called a "tokenized" file, is a condensed version of a BASIC program. It has one-byte "tokens" instead of the ATASCII characters to represent the BASIC commands.

Tokenized programs are moved back and forth between the disk drive and the computer console by SAVE and LOAD commands. Tokenized versions of a file are generally shorter than untokenized versions. For this reason, many programmers prefer to store their final programs in the tokenized form because they will load faster and use less disk space. A tokenized version retains its symbol table from retrieval to retrieval.

LOAD (LO.)

Format: LOAD filespec

Example: LOAD "D1: DOSEX.BAS"

RETURN

This command is used to load a file from a particular diskette in a disk drive into the user program RAM area. To use this command to load a file called DOSEX.BAS, the file (DOSEX.BAS) must have been previously saved using the BASIC command, SAVE. This command only loads a tokenized version of a program.

This command can also be used in "chaining" programs (Figure 5-1). If you have a program that is too big to run in your available RAM, you can use the LOAD command as the last line of the first program (Figure 5-1). Therefore, when the program encounters the LOAD statement, it will automatically read in the next part of the program from the diskette. However, the second program must be able to stand alone without depending on any variables or data in RAM from the first program. The loaded program will not execute until you type **RUN** **RETURN**, at which time the previous program and any variables will be cleared (see RUN for another example).

```
100 REM Chain program
110 LOAD "D1: CHAIN.BAS"
```

Figure 5-1 Example of Program Chaining

SAVE (S.)

Format: SAVE filespec
Example: SAVE "D1: EXAMP2.BAS"

RETURN

This command causes the computer system to save a program on diskette with the filespec name designated in the command. SAVE is the complement of LOAD and stores programs in tokenized form.

LIST (L.)

Formats: LIST filespec ,lineno ,lineno
device
Examples: LIST "D: DATEFIL.LST"
LIST "P:"
LIST "P:", 10, 100

One use of the LIST command in BASIC is very similar to the SAVE command as it can take a program from user program RAM and store it onto a particular drive with any name you want to assign it (illustrated by the first example). However, the program is stored in standard ATASCII text and not as tokens. Differences in the formatting of data storage also allow LIST to be much more flexible than SAVE. As shown in the above format examples, you can specify a single device (e.g., P:, E:, C:, D:, D2:, etc.), or you can specify line numbers to be listed to a designated device (e.g., "P:", 100, 200).

If you do not specify a device after the LIST command, any line numbers you enter will be displayed on the screen. The screen (E:) is always the default device for this command.

In summary, the principle difference between LIST and SAVE is that LIST moves standard ATASCII text to a number of different devices whereas SAVE can only save tokenized BASIC programs on a diskette.

ENTER (E.)

Format: ENTER filespec
Example: ENTER "D: LIST2.LST"

This command causes the computer to move a file on diskette with the referenced filespec into RAM. The program is entered in untokenized form and is interpreted as the data is received. ENTER, unlike LOAD, will not destroy a RAM-resident BASIC program, but will merge the RAM-resident program and the disk file being loaded. If there are duplicate line numbers in the two programs, the line in the program being entered will replace the same line in the RAM-resident program.

RUN

Format: RUN filespec

Example: RUN "D2: MYFILE.BAS"

This command causes the computer to LOAD and RUN the designated filespec. It is a combination of the two commands, LOAD and RUN. However, the RUN command can only be used with tokenized files. Therefore, you cannot execute a RUN "D2: LIST.LST" command.

To chain programs and cause a second segment of a file to load and run automatically, you can use a RUN "D: filespec" as the last line of the first segment. However, the second program must be able to stand alone without depending on any variables or data in RAM from the first program. Before running the first segment, make sure you have saved it on a diskette, as the RUN statement will wipe out your RAM-resident first segment when the second segment is loaded.

INPUT/OUTPUT CONTROL BLOCKS

An I/O operation is controlled by an I/O Control Block (IOCB). An IOCB is a specification of the I/O operation, consisting of the type of I/O, the buffer length, the buffer address, and two more auxiliary control variables of which the second is usually 0. ATARI BASIC sets up eight IOCB's and dedicates three to the following:

- IOCB #0 is used by BASIC for I/O to E:
- IOCB #6 is used by BASIC for I/O to S:
- IOCB #7 is used by BASIC for LPRINT, CLOAD, and SAVE commands.

IOCB's #1 through #5 can be used freely, but the dedicated IOCB's should be avoided unless a program does not make use of one of the dedicated uses mentioned above. IOCB #0 can never be opened or closed from a BASIC program.

IOCB's WITH INPUT/OUTPUT COMMANDS

Each input/output command must have an IOCB associated with it. The I/O commands that can be used in connection with DOS II are the following:

OPEN/CLOSE
INPUT/PRINT
PUT/GET
STATUS
XIO

USING THE OPEN/CLOSE COMMANDS

OPEN (O.)

Format: OPEN #iocb, aexp1, aexp2, filespec

Example: 100 OPEN #2, 8, 0, "D1: ATARI800.BAS"

The OPEN statement links a specific IOCB to the appropriate device handler, initializes any CIO-related control variables (see Glossary), and passes any device-specific options to the device handler. The parameters in this statement are defined in Figure 5-2.

#	Mandatory character entered by user.
iocb	A number between 1 and 7 that refers to a device or file.
aexp1	Number that determines the type of operation to be performed.
Code	4 = input operation; positions file pointer to start of file.
	6 = disk directory input operation.
	8 = output operation; positions file pointer to start of file.
	9 = end-of-file append operation; positions file pointer to end of file.
	Code 9 allows program input from screen editor without user pressing RETURN .
	12 = input and output operation; positions file pointer to start of file.
aexp2	Device-dependent auxiliary code. An 83 (ASCII S) in this position causes the ATARI 820™ Printer to print sideways; otherwise it is always 0 (zero).
filespec	Specific file designation (see Section 1 for filespec definition).

Figure 5-2 Explanation of OPEN Statement Parameters

In the example, OPEN #2, 8, 0, "D1: ATARI800.BAS", IOCB #2 is opened for output to a file on Drive 1 designated as ATARI800.BAS. If there is no file by that name in Drive 1, the DOS creates one. If a file by that name already exists, the OPEN statement destroys that file and creates a new one. If the IOCB has already been opened, the screen displays an ERROR-129 (File Already Opened).

CLOSE (CL.)

Format: CLOSE #iocb
Example: 300 CLOSE #2

The CLOSE command releases the IOCB that had been previously opened for read/write operations. The number following the mandatory # must be the same as the IOCB reference number used in the OPEN statement (see example below). If the IOCB has already been opened to one device and an attempt is made to open the same IOCB to another device without closing, the first ERROR-129 displays on the screen. The same IOCB cannot be used for more than one device at a time. You will not get an error message if you close a file that has already been closed.

```
10 OPEN #1, 8, 0, "D: FIL.BAS"
20 CLOSE #1
```

Figure 5-3 Example of Opening and Closing a File

Note: The END command will close all open files (except IOCB #0).

USING THE INPUT/PRINT COMMANDS

INPUT (I.)

Format: INPUT $\left[\begin{array}{c} \{ \} \\ \{ \} \end{array} \right] \left\{ \begin{array}{c} \{ \text{avar} \} \\ \{ \text{svar} \} \end{array} \right\} \left[\begin{array}{c} \{ \text{avar} \} \\ \{ \text{svar} \} \dots \end{array} \right]$

Examples: 100 INPUT #2; X, Y
100 INPUT #2; N\$

This command is used to request data (either numerical or string) from a specified device. INPUT is the complement of PRINT. When used without a #iocb, the data is assumed to be from the default device (E:). INPUT uses record I/O (see PRINT).

```
5 REM **CREATE DATA FILE**
7 REM **OPEN WITH 8 CREATES DATA FILE**
10 OPEN #1,8,0,"D:WRITE.DAT"
20 DIM WRT$(60)
30 ? "ENTER A SENTENCE NOT MORE THAN 60
CHARACTERS."
35 INPUT WRT$
38 REM **WRITE DATA TO DISKETTE**
40 PRINT #1,WRT$
45 REM **CLOSE DATA FILE**
50 CLOSE #1
55 REM ** OPEN DATA FILE FOR READ**
58 REM **OPEN WITH 4 IS A READ ONLY**
60 OPEN #1,4,0,"D:WRITE.DAT"
65 REM ** READ DATA FROM DISKETTE**
70 INPUT #1,WRT$
75 REM ** PRINT DATA**
80 PRINT WRT$
85 REM ** CLOSE DATA FILE**
90 CLOSE #1
```

Figure 5-4 Sample INPUT/PRINT Program

In Figure 5-4, Line 35 allows the user to type in data on the keyboard (default device). In Line 70, the INPUT statement reads the contents of the string from the opened file.

PRINT (PR. or ?)

Format: PRINT $\left\{ \begin{array}{c} \{ \text{#iocb} \} \\ \{ \text{exp} \} \end{array} \right\} \left[\begin{array}{c} \{ \} \\ \{ \} \end{array} \right] \left[\begin{array}{c} \{ \text{exp} \} \dots \\ \{ \text{exp} \} \end{array} \right]$

Examples: 100 PRINT #2; X, Y
100 PRINT #2; A\$
100 ? C\$
100 PRINT "X = ",X

This command writes an expression (whether string or arithmetic) to the opened device with the same IOCB reference number.

If no IOCB number is specified, the system writes the expression to the screen, which is the default device. If the information is directed to a device that is not open, ERROR-133 displays on the screen.

PRINT performs what is called record I/O. Records are sets of bytes separated by end-of-line characters (9B Hex). The size of a record is arbitrary. Record size can be determined by the length of a string printed to a diskette file or the format of an arithmetic variable. It can also be the length of a string of characters entered from the keyboard and terminated by **RETURN**.

The INPUT statement cannot (generally) read a record that is longer than 110 characters in length. If you PRINT a record to the disk that you will later want to INPUT, it is best to limit the size of the PRINTED records to 110 characters or less.

DIRECT ACCESSING WITH THE NOTE/POINT COMMANDS

NOTE (NO.)

Format: NOTE #iocb, avar, avar

Example: NOTE #2, A, B

Files are created sequentially and are normally accessed from beginning to end. If you want to access the records in a file in a nonsequential manner (directly), you can either read the file sequentially and stop at the record you want; or, you need a special method of addressing the record you want.

```
1 REM NOTEST - NOTE STATEMENT DEMO
2 REM THIS PROGRAM READS LINE OF DATA
3 REM FROM THE KEYBOARD AND STORES
4 REM THEM ON DISK IN FILE D:DATFIL.DAT.

5 REM POINTERS ARE STORED IN D:POINTS.DA
T.
20 DIM A$(40)
25 OPEN #1:8,0,"D:DATFIL.DAT"
27 OPEN #2:8,0,"D:POINTS.DAT"
30 REM READ LINE OF DATA FROM K:
40 INPUT A$
41 LPRINT A$
42 REM IF RETURN ONLY, THEN STOP
45 IF LEN(A$)=0 THEN 100
50 NOTE #1,X,Y
55 REM STORE LINE OF DATA.
60 PRINT #1:A$
61 REM STORE POINTER TO BEGINNING OF
62 REM LINE OF DATA.
65 PRINT #2:X);";Y
70 LPRINT "SECTOR # = ";X;"BYTE # = ";Y
90 GOTO 40
95 REM INDICATE END OF FILE
100 PRINT #2:0);";0
110 END
```

Figure 5-5 Sample NOTE Program

The former is very time-consuming for large files, so DOS II incorporates NOTE and POINT to give you the capability of accessing a file randomly. To get to a record without going through every record that precedes it, you need to let the computer know what record you want. This requires a “note” of the file’s sector, so you use a NOTE command before each write and save the returned value in a table.

This command gets the value of the current file pointer for the file using the specified IOCB. The file pointer specifies the exact position in the file where the next byte is to be read or written. This command stores the absolute disk sector number in the first arithmetic variable and the current byte number in the second. Sector numbers range from 1 to 719 and byte numbers range from 0 to 124. The following program listing and sample run illustrate one way of using NOTE to store keyboard input into a specified file location.

The printout in Figure 5-5 represents a sample of the NOTE program. In the sample run, we used numbers, but you can type any string for A\$ up to 40 characters.

This sample program was run on a diskette that contained the DOS.SYS, DUP.SYS, and MEM.SAV files. Your sector and byte numbers may be different. Our sample entries were 45, 55, 75, 80, 90, 100, 110.

```
45
SECTOR # = 145      BYTE # = 9
55
SECTOR # = 145      BYTE # = 12
75
SECTOR # = 145      BYTE # = 15
80
SECTOR # = 145      BYTE # = 18
90
SECTOR # = 145      BYTE # = 21
100
SECTOR # = 145      BYTE # = 24
110
SECTOR # = 145      BYTE # = 28
```

Figure 5-6 Sample Run of NOTE Program

Figure 5-6 is the sample run that appears on your screen.

POINT (P.)

Format: POINT #iocb, avar, avar

Example: 100 POINT #2, A, B

POINT is the complement of NOTE. This command sets the file pointer to an arbitrary value determined by the arithmetic variables. POINT is used when reading specified file locations (sector and byte) into RAM. The first arithmetic variable specifies (points to) the sector number, and the second arithmetic variable specifies the next byte number into which the next byte will be read or written. As with the NOTE command, the sector number range is between 1 and 719 and the byte range is between 0 and 124. If you point out of an opened file, you will get a File Number

Mismatch error message. The program listing (Figure 5-7) and sample run (Figure 5-8) contain an example of the POINT command to read data created by the program shown as the example for the NOTE command.

When run, this program prints the keyboard input by sector and byte in reverse order from the way it was written to the diskette.

```
1 REM POINTEST - POINT STATEMENT DEMO
2 REM THIS PROGRAM TAKES THE FILE
3 REM CREATED BY NOTEST AND PRINTS
4 REM THE LINES IN REVERSE ORDER.
10 DIM B(20,1)
20 DIM A$(40)
25 REM OPEN DATA FILE
30 OPEN #1,4,0,"D:DATFIL.DAT"
35 REM OPEN POINTER FILE
40 OPEN #2,4,0,"D:POINTS.DAT"
45 REM READ POINTERS INTO AN ARRAY
50 FOR I=0 TO 20
60 INPUT #2;X,Y
70 B(I,0)=X:B(I,1)=Y
80 IF X=0 AND Y=0 THEN LAST=I:I=20
90 NEXT I
95 REM PRINT FILE IN REVERSE ORDER
100 FOR I=LAST-1 TO 0 STEP -1
110 X=B(I,0):Y=B(I,1)
120 POINT #1,X,Y
130 LPRINT "SECTOR # = ";X;"BYTE # = ";Y

140 INPUT #1;A$
150 LPRINT A$
160 NEXT I
```

Figure 5-7 Sample POINT Program


After you type the sample NOTE and POINT programs, type **RUN** 

Figure 5-8 is the sample run.

```
SECTOR # = 145      BYTE # = 28
110
SECTOR # = 145      BYTE # = 24
100
SECTOR # = 145      BYTE # = 21
90
SECTOR # = 145      BYTE # = 18
80
```

```

SECTOR # = 145      BYTE # = 15
75
SECTOR # = 145      BYTE # = 12
55
SECTOR # = 145      BYTE # = 9
45

```

Figure 5-8 Sample RUN of POINT Program

USING THE PUT/GET COMMANDS

PUT (PU.)

Format: PUT #iocb, aexp

Example: 100 PUT #6, ASC ("A")

The PUT command writes a single byte (value from 0-255) to the device specified by the IOCB reference number. In Figure 5-9 the PUT command is used to write each number you type into an array dimensioned as A(50). You can enter up to 50 numbers, each of which should be less than 256. This command is used to create data files or to append data to an existing file.

```

10 GRAPHICS 0:REM PUT/GET DEMO
20 DIM A(50),A$(10)
30 GRAPHICS 0:? " PUT AND GET TO DISK PR
   OGRAM EXAMPLE":?
40 ? "Is this to be a READ or a WRITE?":
   INPUT A$:?
50 IF A$="READ" THEN 170
60 IF A$<>"WRITE" THEN PRINT "?":GOTO 40

70 REM WRITE ROUTINE
80 OPEN #1:8,0,"D1:EXAMPL1.DAT"
90 ? "Enter a number less than 256":INPU
   T X
95 REM **WRITE NUMBER TO FILE**
100 PUT #1,X
110 IF X=0 THEN CLOSE #1:GOTO 130
120 GOTO 90

```

Figure 5-9 Sample PUT Program

GET (GE.)

Format: GET #iocb, avar

Example: 100 GET #2, X

This command reads a single byte from the device specified by the IOCB reference number into the specified variable. The second part of the program example (Figure 5-10) illustrates the GET command. It allows you to retrieve each byte stored by the PUT command.

Note that INPUT/PRINT and GET/PUT are incompatible types of INPUT/OUTPUT. PRINT inserts end-of-line (EOL) characters between records and INPUT uses them to determine a record. GET and PUT merely write single bytes to a file without separating them with EOL. A file created by using PUT statements looks like one large record unless you have placed an EOL (9B Hex) character into the file.

```
130 GRAPHICS 0:?:?:? "Read data in file n
ow?":INPUT A$:?
140 IF A$="NO" THEN END
150 IF A$<>"YES" THEN 130
160 REM READ OUT ROUTINE
170 OPEN #2,4,0,"D1:EXAMPL1.DAT"
180 FOR E=1 TO 50
185 REM **READ NUMBER(S) FROM FILE**
190 GET #2,G:AKE)=G
200 IF G=0 THEN GOTO 230
210 PRINT "BYTE # ";E;"=");G
220 NEXT E
230 CLOSE #2
```

Figure 5-10 Sample GET Program

After you have typed the sample PUT/GET program, type **RUN** .

When you run the program shown in Figure 5-10 it will print the numbers entered from the keyboard together with the byte numbers in which each was stored.

After you type the program, type **RUN**  using number entries 2, 5, 67, 54, 68.

Figure 5-11 is the sample run of the PUT/GET program.

```
BYTE #1=2
BYTE #2=5
BYTE #3=67
BYTE #4=54
BYTE #5=68
```

Figure 5-11 Sample Run of the PUT/GET Program

USING THE STATUS COMMAND

STATUS (ST.)

Format: STATUS #iocb, avar

Example: 100 STATUS #5, ERROR

The STATUS command is used to determine the condition (state) of a file. This command is a CIO command and checks for several ways an error occurs. The first set of possible errors it checks for is as follows:

Sector buffer available?	If no, then ERROR-161
Legal device number?	If no, then ERROR-20
Legal filename?	If no, then ERROR-170
File on diskette?	If no, then ERROR-170
File locked?	If yes, then ERROR-167

You can also identify all I/O serial bus errors with a STATUS command. These are as follows:

Device timeout	ERROR-138
Device not acknowledged	ERROR-139
Serial bus error	ERROR-140
Serial bus data frame overrun	ERROR-141
Serial bus checksum error	ERROR-142
Device done	ERROR-144

To use this command, you must open the file as an input-only file, then close the file. Only then can you issue a STATUS command. It is advisable that you use the XIO command form for this command as it is more reliable and you are able to associate a specific filename with the error you are trying to find.

Figure 5-12 allows you to check the status of your disk drive with a TRAP statement. Before running the program, turn off your disk drive.

```
10 GRAPHICS 0:REM TRAP/STATUS DEMO
20 DIM A(50),A$(10),D$(1)
30 GRAPHICS 0:? " PUT AND GET TO DISK PR
OGRAM EXAMPLE":?
40 ? "Is this to be a READ or a WRITE?":
INPUT A$:?
50 IF A$="READ" THEN 160
60 IF A$<>"WRITE" THEN PRINT "?":GOTO 40

70 REM WRITE ROUTINE
80 TRAP 400:OPEN #1,8,0,"D1:EXAMPL1.DAT"

90 ? "Enter a number less than 256":INPU
T X
100 PUT #1,X
110 IF X=0 THEN CLOSE #1:GOTO 130
120 GOTO 90
130 GRAPHICS 0:? :? "Read data in file n
ow?":INPUT A$:?
140 IF A$="NO" THEN END
150 IF A$<>"YES" THEN 130
160 REM READ OUT ROUTINE
170 TRAP 400:OPEN #1,4,0,"D1:EXAMPL1.DAT
"
180 FOR E=1 TO 50
190 GET #1,G:A(E)=G
200 IF G=0 THEN GOTO 230
210 PRINT "BYTE # ";E;"=";G
220 NEXT E
230 CLOSE #1
240 END
```

```

400 TRAP 40000:STATUS #1,ST:IF ST<>138 A
ND ST<>139 THEN PRINT "HELP":? ST:GOTO 4
30
410 ? "Is your disk drive turned on?"
420 ? "Type Y if you turned on the disk
drive.":INPUT D$
430 CLOSE #1:GOTO 40

```

Figure 5-12 Sample STATUS Program

SUBSTITUTING THE XIO COMMAND FOR DOS MENU OPTIONS

XIO (X.)

Format: XIO cmdno, #iocb, aexp1, aexp2, filespec

Example: 100 XIO 3, #6, 4, 0, "D: TEST.BAS"

The XIO command is a general INPUT/OUTPUT statement used for special operations. It is used when you want to perform some of the functions that would otherwise be performed using the DOS Menu selections. These XIO commands are used to open a file, read or write a record or character, close a file, store status, reference a location in a file for reading or writing, or to rename, delete, lock, or unlock a file. Note that XIO calls need filespecs.

CMDNO (command number) is used to designate just which of the operations is to be performed.

CMDNO	OPERATION	EXAMPLE
3	OPEN	XIO 3, #1, 4, 0, "D: TEST.BAS"
5	GET Record	XIO 5, #1, 0, 0, "D: TEST.BAS"
7	GET Characters	XIO 7, #1, 0, 0, "D: TEST.BAS"
9	PUT Record	XIO 9, #1, 0, 0, "D: TEST.BAS"
11	PUT Characters	XIO 11, #1, 0, 0, "D: TEST.BAS"
12	CLOSE	XIO 12, #1, 0, 0, "D: TEST.BAS"
13	STATUS Request	XIO 13, #1, 0, 0, "D: TEST.BAS"
32	RENAME	XIO 32, #1, 0, 0, "D: OLD, NEW"
33	DELETE	XIO 33, #1, 0, 0, "D: TEMP.BAS"
35	LOCK FILE	XIO 35, #1, 0, 0, "D: ATARI.BAS"
36	UNLOCK FILE	XIO 36, #1, 0, 0, "D: DOSEX.BAS"

Note: Do not use the *device name* twice when renaming a file; i.e. do not use "D: OLD, D: NEW."

PROGRAM EXAMPLE OF XIO COMMAND USES

Figure 5-13 allows you to create a file for each month of the year into which you can enter the names and birthdays of your family and friends. The program uses XIO statements to create a file for each month, to lock and unlock each file as needed by the program, and to close the file when you are through with it.

Line 20 defines the disk file D: BIRTHDAY as FILE\$. Then in Line 170, FILE\$ is opened with an XIO statement for input. The XIO statement in Line 390 unlocks the proper file. The XIO statement in Line 400 creates the file and allows you to write to the file. The next XIO statement, in Line 430, closes the file and the next line's XIO statement locks the file to prevent it from being accidentally overwritten or erased.

```

5 GRAPHICS 0
10 DIM A$(5),D$(15),FILE$(20),DATE$(20),
MON$(20),ERR$(20),NAME$(40)
20 FILE$="D:BIRTHDAY."
30 ERR$="ERROR IN MONTH #"
100 GRAPHICS 0: ? "PLEASE TYPE MONTH NUMB
ER (1-12)"
110 TRAP 100: INPUT MONTH
120 TSTEND=0
130 MONTH=INT(MONTH)
140 IF MONTH<1 OR MONTH>12 THEN ? ERR$:G
OTO 100
145 GOSUB 1000+MONTH
150 FILE$(12)=STR$(MONTH)
160 EOF=0
170 TRAP 700: XIO 3,#2,4,0,FILE$
180 TRAP 600: FOR I=0 TO 1 STEP 0
190 INPUT #2;NAME$
200 INPUT #2;DATE$
210 EOF=EOF+1
220 IF EOF=1 THEN ? "BIRTHDAYS IN ";MON$
; " ARE: " : ?
224 TEMP=LEN(NAME$)
225 NAME$(TEMP+1)=" "
226 NAME$(30)=" "
227 NAME$(TEMP+2,30)=NAME$(TEMP+1)
230 ? NAME$,DATE$
240 NEXT I
299 REM MKE NEW ENTRIES IN BIRTHDAYS
300 ? "WOULD YOU LIKE TO MAKE A NEW BIRT
HDAY ENTRY": INPUT A$
310 IF A$<>"YES" THEN GOTO 20
320 ? "PLEASE TYPE PERSONS NAME"
330 INPUT NAME$
340 ? "PLEASE TYPE PERSONS BIRTHDAY (MM-
DD-YY)"
350 INPUT DATE$
360 MONTH=INT(VAL(DATE$))
370 IF MONTH<1 OR MONTH>12 THEN ? ERR$,D
ATE$:GOTO 300
380 FILE$(12)=STR$(MONTH)
390 TRAP 400: XIO 36,#3,0,0,FILE$: OPEN #2
,9,0,FILE$:GOTO 410
400 CLOSE #2: XIO 3,#2,0,0,FILE$
410 PRINT #2;NAME$
420 PRINT #2;DATE$
430 XIO 12,#2,0,0,FILE$
440 XIO 35,#2,0,0,FILE$
450 GOTO 300
600 CLOSE #2: IF EOF=0 THEN ? "NO BIRTHDA
YS IN ";MON$

```

```

610 MONTH=MONTH+1
620 IF MONTH>12 THEN MONTH=1
630 TSTEND=TSTEND+1
640 IF TSTEND=1 THEN GOTO 145
650 GOTO 300
700 EOF=0:GOTO 600
1001 MON$="JANUARY":RETURN
1002 MON$="FEBRUARY":RETURN
1003 MON$="MARCH":RETURN
1004 MON$="APRIL":RETURN
1005 MON$="MAY":RETURN
1006 MON$="JUNE":RETURN
1007 MON$="JULY":RETURN
1008 MON$="AUGUST":RETURN
1009 MON$="SEPTEMBER":RETURN
1010 MON$="OCTOBER":RETURN
1011 MON$="NOVEMBER":RETURN
1012 MON$="DECEMBER":RETURN
3000 FILE$(12,12+LEN(STR$(MONTH)))=STR$(
MONTH):? FILE$

```

Figure 5-13 Sample XIO Programs

When you run this program, enter a number from 1 to 12. The program will check to see whether or not there are any entries in the file. If there are none, the screen will display the message NO BIRTHDAYS IN followed by whatever month you selected. If you have made entries, the screen will display the names of the people and their birthdays for that month. In either event, the screen will display the names and birthdays for the month you selected and the succeeding month (as a failsafe feature so you will not forget an important birthday that comes at the first part of the next month). When you do not wish to see another file or make another birthday entry, type **NO** to each prompt message and the program will end.

SAVING AND LOADING PROGRAMS AND DATA WITH ATARI BASIC

You cannot modify a program in tokenized form. Therefore, to modify a program which has been saved in tokenized form, the program should first be modified into an untokenized version. This prevents the size of the internal symbol table (see Tokenized and Untokenized Files) from increasing unnecessarily. In Section 3 you used the SAVE and LOAD commands to store and retrieve programs in tokenized form. In that same section, you used the SAVE command to store a program on diskette as D: INTEREST.SAV. What you will do in the example that follows, is to modify INTEREST.SAV by first loading in the tokenized program and then list the program back to diskette in the untokenized version. The untokenized version can then be loaded into the computer and modified.

If you have erased the INTEREST.SAV program from diskette, refer to Figure 5-14 and retype it into your computer.

```

100 REM *** INTEREST
110 PRINT "IF YOU TYPE THE AMOUNT OF PRI
NCIPAL"

```

```

120 PRINT "AND THE INTEREST RATE PER YEA
R, I WILL"
130 PRINT "SHOW YOU HOW YOUR MONEY GROWS
, YEAR BY"
140 PRINT "YEAR. TO STOP ME, PRESS THE B
REAK KEY."
150 PRINT
160 PRINT "PRINCIPAL";
165 INPUT P
170 PRINT "INTEREST RATE";
175 INPUT R
180 LET N=1
190 PRINT
200 LET A=P*(1+R/100)^N
210 PRINT "YEAR = ";N
220 PRINT "AMOUNT = ";A
230 LET N=N+1
240 GOTO 190

```

Figure 5-14 Sample Interest Program

LIST AND ENTER

Below are the steps to store and retrieve this program on diskette using the LIST and ENTER commands. Follow the first set of instructions below if you have an ATARI 810 Disk Drive, and the second set of instructions below if you have an ATARI 815 Dual Disk Drive.

For an ATARI 810 Disk Drive:

1. Use your System Diskette to load DOS II.
2. Remove your System Diskette and put in your data diskette.
3. Type **LOAD "D: INTEREST.SAV"** **RETURN**.
4. Type **LIST** **RETURN** to see D: INTEREST.SAV on the screen.
5. Now type **LIST "D: INTEREST.LIS"** **RETURN** to store the untokenized program on diskette. (Program will not appear on screen.)
6. Type **NEW** **RETURN**. (This eliminates the program INTEREST.SAV in memory.)
7. Type **ENTER "D: INTEREST.LIS"** **RETURN**. (This brings back the program in untokenized form.)
8. Type **LIST** **RETURN**. Using the appropriate keys, change program Line 160 from "Principal" to "Principal Amount."
9. Type **LIST "D: INTEREST.LIS"** **RETURN** to store the change in the program already on diskette.
10. Type **ENTER "D: INTEREST.LIS"** **RETURN**.

For the ATARI 815 Dual Disk Drive:

1. Use your System Diskette in Drive 1 to load DOS II, and place your data diskette in Drive 2.
2. Type **LOAD "D2: INTEREST.SAV"** **RETURN**.
3. Type **LIST** **RETURN** to see D2: INTEREST.SAV on the screen.
4. Now type **LIST "D2: INTEREST.LIS"** **RETURN** to store the untokenized program on diskette. (Program will not appear on screen.)
5. Type **NEW** **RETURN**.
6. Type **ENTER "D2: INTEREST.LIS"** **RETURN**.
7. Use the appropriate keys to change program Line 160 from "Principal" to read "Principal Amount."
8. Type **LIST "D2: INTEREST.LIS"** **RETURN** to store the change in the program already on diskette.
9. Type **ENTER "D2: INTEREST.LIS"** **RETURN**.

To run the program from diskette:

Type **RUN** **RETURN**. The computer retrieves and executes the program (see Figure 5-15). The number entries used for the INPUT statement were 1200 dollars for principal and 12 percent for the interest rate.

```
RUN
IF YOU TYPE THE AMOUNT OF PRINCIPAL
AND THE INTEREST RATE PER YEAR, I WILL
SHOW YOU HOW YOUR MONEY GROWS, YEAR BY
YEAR, TO STOP ME, PRESS THE BREAK KEY.

PRINCIPAL?1200
INTEREST RATE?12

YEAR = 1
AMOUNT =1343.999988

YEAR = 2
AMOUNT =1505.28

YEAR = 3
AMOUNT =1685.913576

STOPPED AT LINE 200
```

Figure 5-15 Sample Run of Interest Program

If you wish to save the original program AND the modified version, give the modified program a new name.

To edit or change a program follow the steps below.

For saving a listed version on diskette:

Type **LIST "D: INTEREST.LIS"** **RETURN** and those specified lines will be merged to the existing program.

To retrieve the listed version on diskette:

1. Type **NEW** **RETURN**. (This deletes the tokenized version and its symbol table from memory.)
2. Type **ENTER "D: INTEREST"** **RETURN** and you will bring the untokenized version into memory.
3. Type **LIST** **RETURN**.

Note: Change the name of the modified program if you wish to retain both the original and revised programs.

You can also use the LIST and ENTER commands to store and retrieve data files you may want to edit. A data file does not contain any commands or instructions. It contains names for an address directory, numbers used for check amounts, etc. However, you must be able to access these data files. For that, you need a program. So before continuing, type in the sample DATA program in Figure 5-16. When you want to stop running the program, just type a zero for the next check number entry.

```
1 REM THIS PROGRAM WRITES A FILE OF CHECK
K NUMBERS AND THEIR AMOUNTS
5 DIM CHECKNAME$(40)
7 REM **OPEN WITH 8 CREATES DATA FILE**
10 OPEN #1,8,0,"D:CHECKS"
20 CHECKAMT=0:CHECKNAME$=""
25 PRINT "CHECK NUMBER";
30 INPUT CHECKNUM
35 IF CHECKNUM=0 THEN 80
38 REM **WRITE DATA TO DISKETTE**
40 PRINT "CHECK AMOUNT";
45 REM **CLOSE DATA FILE**
50 INPUT CHECKAMT
55 REM ** OPEN DATA FILE FOR READ**
58 REM **OPEN WITH 4 IS A READ ONLY**
60 PRINT "WHO WAS CHECK TO";
65 REM ** READ DATA FROM DISKETTE**
70 INPUT CHECKNAME$
75 REM ** PRINT DATA**
80 ERROR- PRINT #1;CHECKNUM>0 THEN PRINT
T: GOTO 20
85 REM ** CLOSE DATA FILE**
90 CLOSE #1
100 CLOSE #1
```

Figure 5-16 Sample Program to Create a Data File

OPEN AND CLOSE

To access the data file, D: CHECKS, you use the BASIC command OPEN. If there is no file by that name on the diskette, the file is automatically created. To store and retrieve data from a program on diskette:

1. Turn on the disk drive.
2. Put a System Diskette into the disk drive.
3. Turn on the computer and boot the system.
4. Type OPEN #1, 8, 0, "D: DATA" **RETURN**. This tells the computer to allow writing to diskette file, DATA, in Drive 1.
5. Type PRINT #1; X; ", "; Y; ", "; Z **RETURN** (X, Y, Z are numbers).
6. Type CLOSE #1 **RETURN** which this tells the computer the file is finished.

Figure 5-17 is a run of the sample program in Figure 5-16. We used the numbers 100, 101, and 102 for the CHECK NUMBER; 12.50, 24.35, and 102.67, for the CHECK AMOUNT; John Smith, George Brown, and Heavy for WHO WAS CHECK TO?

```
READY
RUN
CHECK NUMBER?100
CHECK AMOUNT?12.50
WHO WAS CHECK TO?JOHN SMITH

CHECK NUMBER?101
CHECK AMOUNT?24.35
WHO WAS CHECK TO?GEORGE BROWN

CHECK NUMBER?102
CHECK AMOUNT?102.67
WHO WAS CHECK TO?HEAVY

CHECK NUMBER?0
```

Figure 5-17 Run of Sample Data Program

Figure 5-18 shows you how the information in Figure 5-17 is stored on diskette.

```
100, 12.51, JOHN SMITH
101, 24.35, GEORGE BROWN
102, 102.67, HEAVY
0, 0
```

Figure 5-18 The Information Stored on Diskette

ACCESSING DAMAGED FILES

There are two types of damaged files:

1. The disk directory entry, which contains the file name, directory pointer (points to the first sector of the file), and the number of sectors in the file.
2. The file itself.

Should the DISK DIRECTORY entry be damaged, there is no way to access the file. If the DISK DIRECTORY entry was accidentally deleted, an ERROR 170 (File Not Found) will appear on the screen. If the number of sectors indicated in the DISK DIRECTORY entry do not coincide (are shorter) than the actual number of sectors in the file, an ERROR 164 (File Number Mismatch) will appear on the screen. In this latter case, you may be able to retrieve that portion of the file which falls within the sector range by initiating the Get Byte program in Figure 5-19.

READY

```
10 OPEN #1,4,0,"D:FILE.1"  
20 OPEN #2,8,0,"D:FILE.2"  
25 TRAP 50  
30 GET #1,A  
40 PUT #2,B  
45 GOTO 30  
50 CLOSE #1  
60 CLOSE #2
```

Figure 5-19 Get Byte Program


Let us say File 1 = the damaged file, and File 2 = the recovery file.

Note: You can only read the sectors that fall BEFORE the damaged sector(s). All other sectors after the damage cannot be accessed. As a result, it would be best to COPY the good files on the damaged diskette to a new diskette to avoid any further problems.

If the file itself is damaged, you can also use the Get Byte program which will transfer each good sector from the damaged file into a recovery file.

THE AUTORUN. SYS FILE

When an AUTORUN.SYS file exists on the diskette in Drive 1, that file will automatically be loaded into RAM and executed (if appropriate) every time you boot the system. This entire process is completed before control of the system is returned to you. The AUTORUN.SYS file can be data; it can also be object code that is loaded, but not executed; or, it could be object code that is loaded and then executed as soon as the load is complete.

Figure 5-20 illustrates the use of AUTORUN.SYS to boot up directly into DOS even if a cartridge is present. After execution, AUTORUN.SYS normally returns to the DOS initialization routine. If during your application, it does not return, or if you allow the use of  before the return, the system initialization must be

completed before proceeding. This is done by modifying two Operating System storage locations: COLDST at address 244 (Hex) and BOOT at address 9 (Hex). COLDST should be cleared to 00 and BOOT is set to 01.

The program listed below sets these two locations to the proper value and then jumps indirectly to the start DOS vector.

If you do not have an Assembler Editor cartridge, you can create the equivalent file using BASIC POKE statements and then saving the Binary File in DOS. The list of decimal numbers to be entered is as follows:

Decimal Address	Decimal Codes
15000	162, 00,
15002	142, 68, 02,
15005	232,
15006	134, 09,
15008	108, 10, 00,

When these codes have been entered in BASIC, type **DOS** **RETURN** to save the file using the K. BINARY SAVE selection from the DOS Menu.

Notice that there is no number entered for the INIT parameter. If you turn off your computer and then turn it back on, you should now boot up directly into DOS. To enter BASIC, simply type **B** **RETURN** or press **SYSTEM RESET**.

```

; Autorun Program
;
; Run DOS without going to cartridge.
;
COLDST = $244
BOOT = $09
DOSVEC = $0A
* = 3A98

DOSGO  LDX #0                (HEX CODE)
        STX COLDST          A2 00
        INX                 8E 44 02
        STX BOOT            E8
        STX BOOT            86 09
        JMP (DOSVEC)        6C 0A 00
        * = $2E0            run address at 2E0
        . WORD DOSGO       98 3A
        . END

```

```

SELECT ITEM OR RETURN FOR MENU
K RETURN
SAVE-GIVE FILE, START, END [,INIT,RUN]
AUTORUN.SYS, 3A98, 3AA2,, 3A98
SELECT ITEM OR RETURN FOR MENU

```

Figure 5-20 An AUTORUN.SYS Example for the Advanced User

ADDITIONAL INFORMATION ABOUT THE DISK DRIVE SYSTEM

ATARI DISKETTES

ATARI Diskettes are thin, mylar, circular sheets covered with an oxide similar to that used on cassette tape. Each ATARI Diskette is 5 1/4 inches in diameter and each is sealed in a special black jacket designed to protect it from being bent, scratched, or contaminated.

Each disk drive requires the diskette created specifically to operate with that drive. The ATARI 810 Disk Drive is a single-density drive and the ATARI 815 Dual Disk Drive is a double-density disk drive. The essential difference between double-density diskettes and single-density diskettes is that the double-density recording technique requires a higher quality recording surface so that it can store twice as much data in the same space. Both types of diskettes are manufactured in the same way. The difference is that the double-density diskettes have been pretested to guarantee they will work with the double-density recording techniques. A blank double-density diskette should work as well on a single-density disk drive, but you should not use a blank, single-density diskette on a double-density disk drive.

Once a diskette has been formatted for use with a single-density disk drive it cannot be used in a double-density disk drive unless you reformat the diskette on the double-density drive. The reverse is also true. If you have a system that uses both types of drives and diskettes, be sure that you clearly label each diskette to show the type of drive on which it was formatted.

ATARI 810 DISK DRIVE

The ATARI 810 Disk Drive is a single drive with single-density recording capabilities. It uses standard 5 1/4 inch flexible diskettes: ATARI 810 Master Diskette II (CX8104), ATARI 810 Blank Diskettes (CX8100), ATARI 810/815 Blank Diskettes (CX8202), and ATARI 810 Formatted Diskettes II (CX8111), each of which stores 88K (88 thousand) bytes. The ATARI 810 Disk Drive contains a built-in microprocessor which gives it an automatic stand-by capability. This means the disk drive motor is not in constant operation, but waits to be "told" when it is needed.

ATARI 815 DUAL DISK DRIVE

The ATARI 815 Dual Disk Drive unit contains two drives and uses a double-density recording technique. A sector on a diskette created on this drive can store 256 bytes of data on each sector, which is twice the number of bytes that the ATARI 810 Disk Drive can store in one sector.

Like the ATARI 810 Disk Drive, this dual drive also uses standard 5 1/4 inch flexible diskettes, but can store over 163K bytes on each of its two diskettes. The ATARI 815 Dual Disk Drive has a built-in microprocessor to control the drives. Unlike the ATARI 810 Disk Drive, the ATARI 815 Dual Disk Drive has a built-in power supply. Each of the two drives within the ATARI 815 Disk Drive has its own individual device number.

DISK DRIVE OPERATION

When you insert a diskette into the disk drive, the spindle hole in the center is automatically placed on the drive hub and the diskette is seated. The circular diskette spins within its protective jacket. When you access the diskette, the magnetic head is placed over the read/write surface.

When you store data on a diskette, the disk drive converts the data it receives from the computer console into coded electrical pulses. These pulses magnetize minute areas of the oxide coating of each diskette while the diskette is spinning.

When you retrieve data from a diskette, the disk drive positions the magnetic head so the area of the diskette where the data is stored passes beneath it. The disk drive's microprocessor controls the positioning and timing of the diskette.

APPENDIX A

ALPHABETICAL DIRECTORY OF BASIC RESERVED WORDS USED WITH DISK OPERATIONS

Note: The period is mandatory after all abbreviated keywords.

RESERVED WORD	ABBREVIATION	BRIEF SUMMARY OF BASIC STATEMENT
CLOSE	CL.	I/O statement used to close a disk file at the conclusion of I/O operations.
DOS	DO.	This command causes the menu to be displayed. The menu contains all DOS utility selections. Passes control from cartridge to DOS utilities.
END		Stops program execution, closes files, and turns off sounds. Program may be restarted using CONT. (Note: END may be used more than once in a program.)
ENTER	E.	I/O command used to retrieve a listed program in untokenized (textual) form. If a program or lines are entered when a program is resident in RAM, ENTER will merge the two programs. If you don't want programs merged, type NEW before using ENTER to load a program into RAM.
GET	GE.	Used with disk operation to input a single byte of data into a specified variable from a specified device.
INPUT	I.	This command requests data from a specified device. The default device is E: (Screen Editor).
LIST	L.	This command outputs the untokenized version of a program to a specified device.
LOAD	LO.	I/O command used to retrieve a saved program in tokenized form from a specified device.
NOTE	NO.	This command stores the absolute disk sector number and the current byte number of the file pointer in its two arithmetic variables.
OPEN	O.	Opens the specified file for input or output operations. Determines the type of operations allowed on a file.

RESERVED WORD	ABBREVIATION	BRIEF SUMMARY OF BASIC STATEMENT
POINT	P.	This command is used in setting the file pointer to a specified location (sector and byte) on the diskette.
PRINT	PR. or ?	I/O command causes output from the computer to specified output device in record format.
PUT	PU.	Causes output of a single byte of data; i.e., a character, from the computer to a specified device.
RUN	R.	Both loads and starts execution of designated filespec.
SAVE	S.	I/O statement used to record a tokenized version of a program in a specified file on a specified device.
STATUS	ST.	Calls status routine for specified device.
TRAP	T.	Directs execution to a specified line number in case of a program error, allowing user to maintain control of program and recover from errors.
XIO	X.	General I/O statement used in a program to perform DOS Menu selections and specified I/O commands.

APPENDIX B

NOTATIONS AND TERMINOLOGY USED WITH DOS II

SYSTEM RESET	Press the SYSTEM RESET key on the keyboard.
RETURN	Press the RETURN key on the keyboard.
[]	Brackets. Brackets enclose optional items.
...	Ellipsis. An ellipsis following an item in brackets indicates you can repeat the optional item any number of times, but are not required to do so.
{ }	Braces. Items stacked vertically in braces indicate you have a choice as to which item you want to insert. Select only one to put in your statement or command.
CAPITAL LETTERS	Capital letters are used to indicate commands, statements, and other functions you must type exactly as they appear.
,./:;”	Punctuation marks. These punctuation marks must be typed as shown in the format of a command or statement. However, do not type brackets or braces.
cmdno	Command number. Used in XIO commands.
exp	Expression. In this manual, expressions are divided into three types: arithmetic, logical, and string expressions.
aexp	Arithmetic expression. Generally composed of a variable, function, constant, or two arithmetic expressions separated by an arithmetic operator (aop).
aexp1	Arithmetic expression 1. This arithmetic expression represents the first auxiliary I/O control byte when used in commands such as OPEN.
aexp2	Arithmetic expression 2. This arithmetic expression represents the second auxiliary I/O control byte when used in commands such as OPEN. Usually it is set to 0. If, however, you want to direct the ATARI 820 Printer to print sideways, you would set this arithmetic expression to 83.
filespec	File specification. Usually a string expression that refers to a file and the device where it is located, e.g., “D1:MYPROC.BAS” for a file on Drive 1.
IOCB	Input/Output Control Block (IOCB). An arithmetic expression that evaluates to a number from 1 to 7. The IOCB is used to refer to a device or file. IOCB 0 is reserved for BASIC for the Screen Editor and should only be used if the Screen Editor is not to be used.

lineno	Line number. A constant that identifies a particular program line in a deferred mode BASIC program. A line number can be any integer from 0 through 32767. Line numbering determines the order of program execution.
var	Variable. Any variable. In this manual, variables are classified as arithmetic variables (avar), matrix variables (mvar), or string variables (svar).
avar	Arithmetic variable. A location where a numeric value is stored. Variable names can be from 1 to 120 alphanumeric characters, but must start with an unreversed, uppercase alphabetic character.
mvar	Matrix variable. Also called a subscripted variable. An element of an array or matrix.
svar	String variable. A location where a string of characters may be stored.

APPENDIX C

ERROR MESSAGES AND HOW TO RECOVER

Note: Error messages 2 through 21 should only occur when running a BASIC program.

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
2	Insufficient Memory	You do not have enough memory to store the statement, or to dimension a new string variable. Delete any unused variable names or add more memory. (See Section 11, <i>BASIC Reference Manual</i> for memory conservation.)
3	Value Error	Either the expected positive integer was negative or the value was not within the expected range.
4	Too Many Variables	You have exceeded the maximum number (128) of variable names and must delete any that are no longer applicable. (See Section 11, <i>BASIC Reference Manual</i> .)
5	String Length Error	You have attempted to read from or write into a location past the dimensioned string size or you have used zero as a reference index. Enlarge DIM size. Do not use zero as an index.
6	Out of Data Error	You do not have enough data in your DATA statements for the READ statements.
7	Line Number Greater Than 32767	Check line number references in statements such as GOTO and RESTORE.
8	Input Statement Error	You have attempted to input a non-numeric value into a numeric variable. Check your variable types and/or input data.
9	Array or String DIM Error	The DIM size exceeds 5460 for numeric arrays or 32767 for strings; an array or string was re-dimensioned; reference was made to an un-dimensioned array or string.
11	Floating Point Overflow/Underflow	You have attempted to divide by zero or to refer to a number with an absolute value less than 1E-99, or greater than or equal to 1E-98.
12	Line Not Found	A GOSUB, GOTO, or THEN referenced a non-existent line number.

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
13	No Matching FOR	A NEXT statement was encountered without a matching FOR.
14	Line Too Long Error	You have exceeded the BASIC line processing buffer length.
15	GOSUB or FOR Line Deleted	A NEXT or RETURN statement was encountered and the corresponding FOR or GOSUB was deleted since the last time the program was run.
16	RETURN Error	Check your program for a missing GOSUB statement.
17	Syntax Error	The computer encountered a line with improper syntax. Fix the line.
18	VAL Function Error	The string in a VAL statement is not a numeric string.
19	LOAD Program Too Long	You don't have enough memory to load your program.
20	Device Number Error	You entered a device number that was not between 1 and 7.
21	LOAD File Error	You attempted to load a nonload file, not a BASIC tokenized file. Tokenized files are created with the SAVE command.

Note: The following are input/output errors that result during the use of disk drives, printers, or other accessory devices. Further information is provided with the auxiliary hardware.

128	BREAK Abort	User hit BREAK key during I/O operation. Stops execution.
129	IOCB* Already Open	OPEN statement within a program loop or IOCB already in use for another file or device.
130	Nonexistent Device	<p>You have tried to access a device not in the handler table; i.e., the device is undefined. This error can occur when trying to access the ATARI 850™ Interface Module without running the RS-232-C AUTORUN.SYS file. Another common occurrence of this error is specifying a filename without a device; i.e., "MYFILE" instead of "D:MYFILE."</p> <p>Check your I/O command for the correct device. Then load and initialize the correct handler.</p>

*IOCB refers to Input/Output Control Block

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
131	IOCB Write-Only	You have attempted to read from a file opened for Write-Only. Open the file for read or update (read/write).
132	Illegal Handler Command	This is a CIO error code. The common code passed to the device handler is illegal. The command is either ≤ 2 or is a special command to a handler that hasn't implemented any special commands. Check your XIO or IOCB command code for an illegal command code.
133	Device/File Not Open	You have not opened this file or device. Check your OPEN statement or file I/O statement for a wrong file specification.
134	Bad IOCB Number	You have tried to use an illegal IOCB index. For BASIC, the range is 1-7 as BASIC does not allow use of IOCB 0. The Assembler Editor cartridge requires the IOCB index to be a multiple of 16 and less than 128.
135	IOCB Write-Only Error	You have tried to write to a device or file that is open for Read-Only. Open the file for write or update (read/write).
136	End of File	Your input file is at end of file. No more data in file.
137	Truncated Record	This error typically occurs when the record you are reading is larger than the maximum record size specified in the call to CIO. (BASIC's maximum record size is 119 bytes.) When trying to use an INPUT (record-oriented type of command on a file that was created with PUT (byte-oriented) commands, results in this problem.
138	Device Timeout	<p>When you sent a command over the serial bus, the device did not respond within the period set by the O.S. for that device command. Either the device number is wrong or the user specified the wrong device; the device is not there (wrong spec); it is unable to respond within the proper period; or it is not connected. If the device is a cassette, the tape baud rate may have been mismeasured or the tape improperly positioned.</p> <p>Examine all connections to make sure they are secure and check the disk drive to make sure it is turned on and set for the correct drive number. Check your command for the correct drive number. Retry the command. If this error recurs, have the disk drive checked.</p>

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
139	Device NAK	The device cannot respond because of bad parameters such as an unaddressable sector. The device might also have received a garbled or illegal command or received improper data from the computer. Check your I/O command for illegal parameters and retry the command. Also check your I/O cables. This is a device specific error, so refer to the documentation for that device.
140	Serial Frame Error	<p>Bit 7 of SKSTAT in the POKEY chip is set. This means that communication from the device to the computer is garbled.</p> <p>This is a very rare error and it is fatal. If it occurs more than once, have your device or computer checked. You can also remove the peripherals one at a time to isolate the problem. For cassettes, try the recovery suggested in Error 138.</p>
141	Cursor Out of Range	Your cursor is out of range for the particular graphics mode you chose. Change the pixel parameters.
142	Serial Bus Overrun	Bit 5 of SKSTAT in POKEY is set. The computer did not respond fast enough to a serial bus input interrupt or POKEY received a second 8-bit word on the serial bus before the computer could process the previous word. This is a rare error. If it occurs more than once, have your computer serviced.
143	Checksum Error	The communications on the serial bus are garbled. The checksum sent by the device is not the same as that calculated for the frame received by the computer. There is no standard recovery procedure because it could be either a hardware or software problem.
144	Device Done Error	The device is unable to execute a valid command. You have either tried to write to a write-protected diskette or device, or the disk drive is unable to read/write to the requested sector. Remove the write-protect tab or turn off the write-protect switch. See specific manuals for other devices.
145	Illegal Screen Mode	You have tried to open the Screen Editor with an illegal graphics mode number. Check your graphics mode call or the aux2 byte in the IOCB.

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
146	Function Not Implemented	The handler does not contain the function; e.g., trying to PUT to the keyboard or issuing special commands to the keyboard. Check your I/O command for the right command and the correct device.
147	Insufficient RAM	Not enough RAM for the graphics mode you selected. Add more memory or use a graphics mode that doesn't require as much memory.
160	Drive Number Error	You specified a drive number that was not 1-8, did not allocate a buffer for the drive, or your drive was not powered up at boot time. Refer to Section 1 of this manual. Check your filespec or byte 1802 for number of drive buffers allocated.
161	Too Many OPEN Files	You don't have any free sector buffers to use on another file. Check Location 1801 for the number of allocated sector buffers. Also make sure no files are open that should not be open.
162	Disk Full	You don't have any more free sectors on this diskette. Use a different diskette that has free sectors.
163	Unrecoverable System I/O Error	This error means that the File Manager has a bug in it. Your DOS or the diskette may be bad. Try using other DOS.
164	File Number Mismatch	The structure of the file is damaged or POINT values are wrong. One of the file links points to a sector allocated to another file. Turn the system off and retry program execution. If this fails, you have lost the file. Try to recover the other files on diskette, then reformat the diskette.
165	File Name Error	Your file specification has illegal characters in it. Check filespec and remove illegal characters.
166	POINT Data Length Error	The byte count in the POINT call was greater than 125 (single density) or 253 (double density). Check the parameters in your POINT statement.
167	File Locked	You have tried to access a locked file for purposes other than reading it. Use DOS Menu option G to unlock the file and retry your command.
168	Device Command Invalid	You issued an illegal command to the device software interface. Check the documentation for that device and retry the command.

ERROR NO.	ERROR NAME	CAUSE AND RECOVERY
169	Directory Full	You have used all the space allocated for the Directory.
170	File Not Found	You have tried to access a file that doesn't exist in the diskette's Directory. Use DOS Menu option A to check the correct spelling of the filename and to be sure it is on the diskette you are accessing.
171	POINT Invalid	You have tried to POINT to a byte in a file not opened for UPDATE. Check the parameters of your OPEN statement or aux1 byte of the IOCB used to open the file.
172	Illegal Append	You have tried to open a DOS I file for append using DOS II. DOS II cannot append to DOS I files. COPY the DOS I file to a DOS II diskette using DOS II.
173	Bad Sectors at Format Time	The disk drive has found bad sectors while formatting a diskette. Use another diskette, as you cannot format a diskette with bad sectors. If this error occurs with more than one diskette, your disk drive may need repair.

APPENDIX D

DOS II MEMORY MAP FOR 32K RAM SYSTEM

Decimal	ADDRESS		CONTENTS
	Hexadecimal		
65535	FFFF	OPERATING SYSTEM	
49152	C000		
49151	BFFF	CARTRIDGES	
32768	8000		
32767	7FFF	SCREEN DISPLAY AREA	
varies			
varies		USER PROGRAM AREA	HIMEM**
varies			LOMEM*
	$2S \left\{ \begin{array}{l} 3305 \\ 1D7C \end{array} \right. \quad 2D \left\{ \begin{array}{l} 3A38 \\ 247C \end{array} \right.$	S B 7 S B 6 S B 5 S B 4 S B 3	DISK UTILITY PROGRAMS
		Sector Buffer 2 Sector Buffer 1 Drive 4 Buffer Drive 3 Buffer Drive 2 Buffer Drive 1 Buffer	BUFFER AREA RESERVED FOR DOS II
6781	1A7D		
6780	1A7C	MINI-DOS (RAM RESIDENT PORTION OF DUP)	
5440	1540		
5439	153F	FILE MANAGEMENT SUBSYSTEM	
1792	0700		
1791	06FF 0600	USER RAM	
	05FF 0000	OPERATING SYSTEM RAM	
0			

* Varies with the number of Drive and Sector Buffers reserved.

**Depends on which Graphics Mode is currently in use.

Note 1: For given Drive Buffer allocation and Sector Buffer allocation, LOMEM can be determined by PEEK Locations 2E7 (LOW) and 2E8 (HIGH) Hex or 743 (LOW) and 744 (HIGH) Decimal.

Note 2: To determine the amount of User Program Area available or HIMEM, you can either make use of the BASIC FRE(O) instruction or PEEK Locations 2E5 (LOW) and 2E6 (HIGH) Hex or 741 (LOW) and 742 (HIGH) Decimal.

APPENDIX E

HEXADECIMAL TO DECIMAL CONVERSION TABLE

FOUR HEX DIGITS							
4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	206	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

*Use this table to convert up to four hex digits.

For example, to convert the hex number 1234 to decimal, add the entries from each of the four columns in the table. For 1, use the column number 4, and so on.

$$\begin{array}{r} 1234 \text{ hex.} = \\ \quad 4096 \\ \quad + 512 \\ \quad + 48 \\ \quad + 4 \\ \hline 4660 \text{ dec.} \end{array}$$

Other examples:

$$\begin{array}{r} EEDD \text{ hex.} = \\ \quad 57344 \\ \quad + 3584 \\ \quad + 208 \\ \quad + 13 \\ \hline 61149 \text{ dec.} \end{array}$$

$$\begin{array}{r} AB \text{ hex.} = \\ \quad 160 \\ \quad + 11 \\ \hline 171 \text{ dec.} \end{array}$$

APPENDIX F

HOW TO SPEED UP DATA TRANSFERS TO DISK DRIVE

Your new DOS II Version 2.05 has the ability to Write with Read Verify (a safety technique that should be used whenever improved reliability is more important than rapid data transmissions). This is the way your DOS II Master Diskette is shipped to you. To save time, however, the information can be written to the diskette without a Read Verify. Memory Location 1913 (decimal) contains the data that determines whether the File Management Subsystem will use Write with Read Verify (50 hex, 80 decimal) or Write without Read Verify (57 hex, 87 decimal). Write without Read Verify is of course faster, but may not be as reliable. To customize your version of DOS II from BASIC, you need to:

```
POKE 1913,80
```

for fast Write (Write without Read Verify). If you would rather have the Write with Read Verify,

```
POKE 1913,87
```

To alter the version of DOS stored on diskette so that your custom version will always boot in, simply type **DOS** and then use an "H" command (WRITE DOS FILES) from the Disk Operating System Menu to store the new version of DOS from RAM onto your diskette.

APPENDIX G

HOW TO INCREASE USER RAM SPACE

This section explains how to change the RAM location number to reflect the number of drives attached to your ATARI Personal Computer System. You will need to poke the correct drive number code into RAM Location 1802 (decimal). The following table gives the correct entries for the number of ATARI 810 Disk Drives. Note that a maximum of four ATARI 810 Disk Drives can be used with DOS II since the switches on each drive can only be set from 1 to 4. Note also that in the Binary Drive Code, there is a 1 corresponding to each drive in the system. Although earlier in this manual, we suggested you designate the ATARI 815 Dual Disk Drive as Drives 1 and 2, and the ATARI 810 DISK DRIVE as Drive 3, you can see by the following listing that there are other options.

CODES FOR NUMBER OF ATARI 810 DISK DRIVES ATTACHED

Drives Allocated	Decimal Drive Code	Binary Drive Code
Drive 1	01	00000001
Drive 2	02	00000010
Drive 3	04	00000100
Drive 4	08	00001000
Drives 1 + 2	03	00000011 (default)
Drives 1 + 3	05	00000101
Drives 1 + 2 + 3 + 4	15	00001111

If your system includes both ATARI 810 and ATARI 815 Disk Drives, you will have to use your ATARI 810 Disk Drives in positions 1 through 4. The dual disk drives can be switch-selected to any of the following pairs: (1 + 2), (3 + 4), (5 + 6), or (7 + 8). The following table gives the correct entries for the number of dual disk drives. This data is also stored in RAM Location 1802 (decimal).

Drives 1 + 2	03	00000011
Drives 3 + 4	12	00001100
Drives 5 + 6	48	00110000
Drives 7 + 8	192	11000000

Example 1.

If you have one ATARI 810 Disk Drive and one ATARI 815 Dual Disk Drive, how should you structure your DOS? Since you must have a Drive 1, you can either set the ATARI 815 to Drives 1 and 2 and the ATARI 810 to Drive 3, or set the ATARI 810 to Drive 1 and the ATARI 815 to Drives 3 and 4. Suppose you choose the former—the ATARI 815 as Drives 1 and 2. This equals a code of 03 (see above table). Then, placing the ATARI 810 in Drive 3 position would equal a code of 04. The value you would enter into RAM Location 1802 would be $03 + 04 = 07$ in decimal, which is 0000111 in binary.

Example 2.

Assume that you have three ATARI 810 Disk Drives and two ATARI 815 Dual Disk Drives. Since the ATARI 810 Disk Drives can only be used in positions 1 through 4, you can see from the tables above that you would have codes of 01, 02, and 03. Placing the dual disk drives in positions 5 + 6 and 7 + 8, you would have codes of 48 and 192. Get the total value by adding these decimal numbers, i.e., $01 + 02 + 03 + 48 + 192 = 246$. This value of 246 is the value you would enter into RAM Location 1802. A 246 is 11110110 in binary.

APPENDIX H

MAJOR DIFFERENCES BETWEEN DOS I (9/24/79) AND DOS II

DOS I	DOS II
Boot time was 11 seconds	Boot time is 7 seconds
Load time for the disk utility package was 0 as it was loaded during the boot.	Load time for DUP is 9 seconds
No Wild Card copy available	Wild Card available enabling you to copy ALL files from one diskette to another.
No MEM. SAV available	MEM. SAV available allowing the user to have more memory space. This makes for a more powerful DOS.
Requires a 1 sector boot	Requires a 3 sector boot enabling DOS II to handle double-density drives. As a result, DOS I and DOS II diskettes cannot be interchanged.
No AUTORUN. SYS	Has an AUTORUN.SYS file enabling a file to be loaded and executed upon booting.
Files copied or duplicated in small buffer	Entire file(s) copied or duplicated into buffer which can be as large as user memory area.
Could not append two files together	SAVE BINARY FILE has "/A" option allowing two files to be appended together.
Did not have a load-and-go	Can create load-and-go type file which enables you to select a file and have it automatically run without entering a RUN address.
Bad sectors on diskette not indicated during formatting.	A diskette with bad sectors cannot be formatted.
Margins not reset automatically	Margins are reset to original position each time DOS II is entered.
One version of DOS I: system with single-density disk drives	Two versions of DOS II: systems with single/double-density disk drives or systems with single-density disk drives only.
Had to redisplay menu before reissuing new command	Choice of redisplay of menu or entering a new command
Could only write DOS File to Drive 1	Can write DOS Files to any drive

DOS I

Could only have three files open simultaneously

NOTE/POINT not available for random access.

DOS II

Can have up to eight files open simultaneously

NOTE/POINT is available for random access.

APPENDIX I

STRUCTURE OF A COMPOUND FILE


COMPOUND FILE STRUCTURE USING C. COPY FILE WITH APPEND

Byte No.	Decimal No.	Hex No.	Description
1	255	FF	Identification Code (PART 1)
2	255	FF	
3	0	00	Starting Address (PART 1)
4	80	50	
5	31	1F	Ending Address (PART 1)
6	80	50	
.	.	.	DATA (PART 1)
.	.	.	32 Bytes
.	.	.	
38	255	FF	Identification Code (PART 2)
39	255	FF	
40	32	20	Starting Address (PART 2)
41	80	50	
42	143	8F	Ending Address (PART 2)
43	80	50	
.	.	.	DATA (PART 2)
.	.	.	112 Bytes
.	.	.	

COMPOUND FILE STRUCTURE USING K. BINARY SAVE WITH APPEND

Byte No.	Decimal No.	Hex No.	Description
1	255	FF	Identifier Code
2	255	FF	
3	00	00	Starting address (PART 1)
4	80	50	
5	31	1F	Ending address (PART 1)
6	80	50	
.	.	.	Data (PART 1)
.	.	.	32 Bytes
.	.	.	
38	32	20	Starting address (PART 2)
39	80	50	
40	143	8F	Ending address (PART 2)
41	80	50	
.	.	.	Data (PART 2)
.	.	.	112 Bytes
.	.	.	

GLOSSARY OF TERMS

Access:	The method (or order) in which information is read from, or written to diskette.
Address:	A location in memory, usually specified by a two-byte number in hexadecimal or decimal format. (Maximum range is 0-FFFF hexadecimal.)
Alphanumeric:	The capital letters A-Z and the numbers 0-9, and/or combinations of letters and numbers. Specifically excludes graphics symbols, punctuation marks, and other special characters.
Array:	A one-dimensional set of elements that can be referenced one at a time or as a complete list by using the array variable name and one subscript. Thus the array B, element number 10 would be referred to as B(10). Note that string arrays are not supported by BASIC, but you can pick up each element within a string; for example, A\$(10). All arrays must be dimensioned before use. A matrix is a two-dimensional array.
ATASCII:	The method of coding used to store text data. In ATASCII (which is a modified version of ASCII, the American Standard Code for Information Interchange), each character and graphics symbol, as well as most of the control keys, has a number assigned to represent it. The number is a one-byte code (decimal 0-255). See the <i>ATARI BASIC Reference Manual</i> for table.
AUTORUN.SYS:	Filename reserved by Disk Operating System.
Baud Rate:	Signaling speed or speed of information interchange in bits per second.
Binary Load:	Loading a binary machine-language object file into the computer memory.
Binary Save:	Saving a binary machine-language object file onto a disk drive or program recorder.
Bit:	Abbreviation of "binary digit." The smallest unit of information, represented by the value 0 or 1.
Boot:	This is the initialization program that "sets up" the computer when it is powered up. At conclusion of the boot (or after "booting up"), the computer is capable of loading and executing higher level programs.
Break:	To interrupt execution of a program. Pressing the  key causes a break in execution.

Buffer:	A temporary storage area in RAM used to hold data for further processing, input/output operations, and the like.
Byte:	Eight bits. A byte can represent one character. A byte has a range of 0–255 (decimal).
CIO:	Central Input/Output Subsystem. The part of the OS that handles input/output.
CLOSE:	To terminate access to a disk file. Before further access to the file, it must be opened again. See OPEN.
Data:	Information of any kind, usually a set of bytes.
Debug:	To isolate and eliminate errors from a program.
Decimal:	A number base system using the digits 0 through 9. Decimal numbers are stored in binary coded decimal format in the computer. See Bit, Hexadecimal, and Octal.
Default:	A condition or value that exists or is caused to exist by the computer until it is told to do something else. For example, the computer defaults to GRAPHICS 0 until another graphics mode is entered.
Delimiter:	A character that marks the start or finish of a data item but is not part of the data. For example, the quotation marks (") are used by most BASIC systems to delimit strings.
Density:	The closeness of space distribution on a storage medium; i.e., the number of bytes per sector. Single density records 128 bytes per sector and double density records 256 bytes per sector.
Destination:	The device or address that receives data during an exchange of information (and especially an I/O exchange). See Source.
Directory:	A summary of files contained on a diskette by filename and file size.
Diskette:	A small disk. A record/playback medium like tape, but made in the shape of a flat disk and placed in an envelope for protection. The access time for a diskette is much faster than for tape.
DOS:	Disk Operating System abbreviation. The software or programs that facilitate use of a disk drive system.
DOS.SYS:	Filename reserved by Disk Operating System.
Drive Specification or Drivespec:	Part of the filespec that tells the computer which disk drive to access. If this is omitted the computer will assume Drive 1.
Drive Number:	An integer from 1 to 8 that specifies the drive to be used.
End of File:	A marker that tells the computer that the end of a certain file on disk has been reached.

Entry Point:	The address where execution of a machine-language program or routine is to begin. Also called the transfer address.
File:	An organized collection of related data. A file is the largest grouping of information that can be addressed with a single name. For example, a BASIC program is stored on diskette as a particular file, and may be addressed by the statements SAVE or LOAD (among others).
Filename:	The alphanumeric characters used to identify a file. A total of eight numbers and/or letters may be used, the first must be a letter.
Filename Extender or Extension:	From 0 to 3 additional characters used following a period (required if the extender is used) after the filename. For example, in the filename PHONLIST.BAS the letters "BAS" comprise the extender.
File Pointer:	A pointer to a location in a file. Each file has its own pointer.
Filespec:	Abbreviation for file specification. A sequence of characters which specifies a particular device and filename. Do not create two files with exactly the same filespec. If you do, and they are both stored on the same diskette, you will not be able to access the second file. And, as they both have the same filespec, the DOS functions of DELETE FILE, RENAME FILE, and COPY FILE will act on both files.
Format:	To organize a new or magnetically (bulk) erased diskette onto tracks and sectors. When formatted, each diskette contains 40 circular tracks, with 18 sectors per track. Each sector can store up to 128 bytes of data (single density) or 256 bytes of data (double density).
Hexadecimal or Hex:	Number base system using 16 alphanumeric characters: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, and F.
Indexed Addressing:	See Random Accessing.
IOCB:	Input/Output Control Block. A section of RAM reserved for addressing an input or output device and processing data received from it or for addressing and transferring data to an output device.
iocb:	An arithmetic expression that evaluates to a number between 1 and 7. This number is used to refer to a device or file.
Input:	To transfer data from outside the computer (say, from a diskette file) into RAM. Output is the opposite, and the two words are often used together to describe data transfer operations: Input/Output or just I/O. Note that the reference point is always the computer. Output always means away from the computer, while Input means into the computer.
INPUT:	A BASIC command used to request either numeric or string data from a specified device.
Kilobyte or K:	1024 bytes. Thus a 16K RAM capacity actually gives us 16 times 1024 or 16,384 bytes.

Least Significant Byte:	The byte in the rightmost position in a number or a word.
Machine Language:	The instruction set for the particular microprocessor chip used, which in ATARI's case is the 6502.
Most Significant Byte:	The byte in the leftmost position in a number or a word.
Null String:	A string consisting of no character whatever. For example, A\$="" stores the null string as A\$.
Object Code:	Machine language derived from "source code," typically from assembly language.
Octal:	The octal numbering system uses the digits 0 through 7. Address and byte values are sometimes given in octal form.
OPEN:	To prepare a file for access by specifying whether an input or output operation will be conducted, and specifying the filespec.
Parameter:	Variables in a command or function.
Peripheral:	An I/O device.
POKEY:	A custom I/O chip that manages communication on the serial bus.
Random Accessing:	The method of reading data from a diskette directly from the byte and sector where it was stored without having to read the entire file sequentially.
Record:	A block of data, delimited by EOL (End-of-Line, 9B Hex) characters.
Sector:	A sector is the smallest block of data that can be written to a disk file or read from a file. Each single-density sector can store 128 bytes of data and each double-density sector can store 256 bytes of data.
Sequential Addressing:	The method of reading each byte from the diskette in order, starting from the first byte in the sector.
Source:	The device or address that contains the data to be sent to a destination. See Destination.
Source Code:	A series of instructions, written in a language other than machine language, which requires translation in order to be executed.
String:	A sequence of letters, characters, stored in a string variable. The string variables name must end with a \$.
System Diskette:	An exact copy of original Master Diskette. Always use backup copies of your Master Diskette instead of the original. Keep backup copies of all important data and program diskettes.
Tokenizing:	The process of interpreting textual BASIC source code and converting it to the internal format used by the BASIC interpreter.

Track:	A circle on a diskette used for magnetic storage of data. Each track has 18 sectors, each with 128 byte storage capability. There are a total of 40 tracks on each diskette.
Variable:	A variable may be thought of as a box in which a value may be stored. Such values are typically numbers and strings.
Write-Protect:	A method of preventing the disk drive from writing on a diskette. ATARI diskettes are write-protected by covering a notch on the diskette cover with a small sticker.

INDEX

A

Access 65, 66, 93
Accessing damaged files 67
Address 93
Additional disk drives 2
Aexp 73
Aexp1 52, 73
Aexp2 52, 73
Alphanumeric 93
Appendices 71-93
Array 93
ATASCII 93
AUTORUN. SYS 9, 67, 68, 93
Avar 74

B

BASIC commands 49
BASIC error messages
BASIC words used
Baud rate 93
Binary load 7, 9, 43, 93
Binary save 7, 9, 39-42, 93
Bit 93
Boot 93
Boot errors 22
Break 93
Buffer 94
Byte 94

C

CIO 94
CLOSE 52, 94
Cmdno 60, 73
Commands
Compound file structure 41, 42
Control blocks 51, 95
Copy file 5, 9, 30, 31
Create MEM.SAV 44
Creating a system diskette 12
 Using an ATARI 810 Disk Drive 12
 Using an ATARI 815 Dual Disk Drive 12, 13

D

Daisy-chaining 2
Data 94

Data files 19, 65
Data transfer to disk drives 37
Debug 94
Decimal 81, 83, 94
Default 94
Delete file 5, 9, 10, 32, 33
Delimiter 94
Density 94
Destination drive 94
Differences between DOS I and DOS II 89, 90
Direct accessing 54
Directory 94
Disk directory 5, 27, 29, 67
Disk drives 1, 2, 3, 11, 69, 87-88
Diskettes 3, 69, 94
Disk Operating System (DOS) 1, 94
DOS command 71
DOS menu 4, 27
DOS menu options 5
DOS.SYS 9, 94
Double density 4, 15, 16, 69, 91
Drive codes 2, 3
Drive number 2, 3, 13, 94
Duplicate file 6, 9, 47, 48
Duplicating a diskette 6, 9, 12, 37
 Using a single disk drive 11, 12, 37
 Using multiple disk drives 11, 13, 38
DUP.SYS 9, 30

E

END command 71
End of file 94
ENTER command 50, 63, 65, 71
Entry point 95
Error code messages 75-80
Exp 73

F

File 95
File Management Subsystem (FMS) 9, 11
Filename 19, 20, 28, 95
Filename extenders 19, 20, 28, 95
File pointer 95
Filespec 19, 20, 28, 52, 73, 95
Formatting a diskette 6, 9-11, 36, 95

G

GET command 57, 58, 71
GET/BYTE program 67
Glossary of terms 93

H

Hexadecimal 81, 83, 95

I

Identifying diskette files 19
Increasing user RAM space 87
Input 95
INPUT/OUTPUT commands 51, 53, 54, 71, 95
IOCB 51, 73, 95
#iocb 52, 95

K

Kilobyte 95

L

Labeling disk drives 3
Labeling diskettes 15
Least significant byte 96
Linenos 74
LIST command 50, 63, 65, 71
LOAD command 23, 25, 49, 71
Lock file 6, 9, 34, 35

M

Machine language 96
Master diskette 9, 11, 12
Memory maps 81
MEM.SAV 6, 9, 29, 30, 44-46
 Creating MEM.SAV 44
 MEM.SAV to load binary files 46
 MEM.SAV to write assembly language programs 45
Menu 4
Most significant byte 96
Mvar 74

N

Notes and terminology 73
NOTE command 54, 71
Null string 96

O

Object codes 96
Octal 96
OPEN command 51, 52, 71, 96

P

Parameters 21, 28, 39, 40, 96
Peripheral 96
POINT command 55, 56, 57, 72
PRINT command 53, 54, 72
Program files 19
PUT command 57, 72

R

RAM 1, 87
Record 96
Rename file 6, 9, 32, 33
Run at address 7, 9, 44, 46
Run cartridge 5, 9, 29, 46
RUN command 24, 25, 51, 72

S

SAVE command 23, 24, 50, 72
Sector 10, 11, 96
Sequential addressing 96
Single density 4, 11, 15, 16, 69
 Conversion of single density to double density 69, 91
Source code 96
STATUS command 58, 59, 72
String 96
Svar 74
System diskette 12, 13, 96

T

Tokenized file 49, 96
Track 10, 96
TRAP command 72

U

Unlock file 6, 9, 35
Untokenized file 49, 71

V

Variable 74, 96

W

Wild cards 21
Write DOS file 30, 35, 36
Write new DOS 6
Write-protecting a diskette 14, 15, 35, 36, 96

X

XIO command 60-62, 72

ERROR CODE NO	ERROR CODE MESSAGE
2	Insufficient Memory
3	Value Error
4	Too Many Variables
5	String Length Error
6	Out of Data Error
7	Number Greater Than 32767
8	Input Statement Error
9	Array or String DIM Error
11	Floating Point Overflow/Underflow Error
12	Line Not Found
13	No Matching FOR Statement
15	GOSUB or FOR Line Deleted
16	RETURN Error
17	Garbage Error
18	Invalid String Character
19	LOAD Program Too Long
20	Device Number Larger Than 7 or Equal to 0
21	LOAD File Error
128	BREAK Abort
129	IOCB* Already Open
130	Nonexistent Device Specified
131	IOCB Write-Only
132	Invalid Command
133	Device or File Not Open
134	Bad IOCB Number
135	IOCB Read-Only Error
136	End of File
137	Truncated Record
138	Device Timeout
139	Device NAK (not acknowledged)
141	Cursor Out of Range for Particular Graphics Mode
142	Serial Bus Data Frame Overrun
143	Serial Bus Data Frame Checksum Error
144	Device Done Error (invalid "done" byte)
145	Read-After-Write Compare Error
146	Function Not Implemented in Handler
147	Insufficient RAM for Selected Graphics Mode
160	Drive Number Error
161	Too Many OPEN Files (no sector buffer available)
162	Disk Full (no free sectors)
163	Unrecoverable System Data I/O Error
164	File Number Mismatch (sector links on disk are messed up)
165	File Name Error
166	POINT Data Length Error
167	File Locked
168	Command Invalid (special operation code)
169	Directory Full (64 files)
170	File Not Found
171	POINT Invalid
172	Attempt to Append to DOS I File Using DOS II
173	Bad Sectors at Format Time

*IOCB refers to Input/Output Control Block


Every effort has been made to ensure the accuracy of the product documentation in this manual. However, because Atari, Inc. is constantly improving and updating the computer software and hardware, we are unable to guarantee the accuracy of the printed material after the date of publication and disclaim liability for changes, errors or omissions.

No reproduction of this document or any portion of its contents is allowed without specific written permission of Atari, Inc., Sunnyvale, CA 94086.

© 1982, ATARI, INC.
ALL RIGHTS RESERVED



PRINTED IN SINGAPORE C061531 REV. A

 A Warner Communications Company