FIRST AND FINEST

OS/A+

OS/A+
OS/A+
OS/A+
OS/A+

Systems Software for
Apple and Atari Computers

OSS PRECISION SOFTWARE™

a reference manual for

O S / A +

an Operating System for Atari Computers +
an Operating System for Advanced users

This manual revised June, 1983

408-446-3099

PREFACE

OS/A+ is the result of the efforts of several persons, and we believe that proper credit should be given. The original Apple version of the console processor (CP) and the original version ("version 2") of the File Manager System (which is, of course, identical with Atari's DOS 2.ØS) were written by Paul Laughton, ex of Shepardson Microsystems, Inc., who also authored the original Apple DOS (version 3.1). The current versions of all other portions are primarily the work of Mark Rose, of OSS, with the collaboration of Bill Wilkinson and Mike Peters.

We realize that OS/A+ is not the most sophisticated, most complete, operating system for any and all microcomputers, but we believe that the inherent power and flexibility that it exhibits within its compact size are a good match for the size and features of the machines it is intended for.

TRADEMARKS

The following trademarked names are used in various places within this manual, and credit is hereby given:

OS/A+, BASIC A+, MAC/65, and C/65 are trademarks of Optimized Systems Software, Inc.

Atari, Atari 4ØØ, Atari 8ØØ, Atari Home Computers, and Atari 85Ø Interface Module are trademarks of Atari, Inc., Sunnyvale, CA.

TABLE OF CONTENTS

TABLE of CONTENTS (cont)

TABLE of CONTENTS (cont)

INTRINSIC COMMAND SUMMARY

CAR	Transfer control to a cartridge
DIR [Dn:][file-name] [output file-spec]	View the disk directory
END	Stop batch execution from within an execute file
ERA [Dn:]file-name	Remove files from a disk
LOAD [Dn:]file-name	Load disk files into memory
NOScreen	Turn off command echo to screen during batch
PRO [Dn:]file-name	Protect files from accidental erasure, writing, or renaming
REM any characters	Print remarks to the screen during batch execution
REN from-file-name to-file-name	Rename a file to a new name
RUN [hex-address]	Transfer control to an address in memory
SAVE file-spec start-address end-address	Save a portion of memory to a disk file
SCR	Cause batch commands to be echoed to the screen
TYP [Dn:]file-name [output-file]	Type a text file to the screen or another file
UNP [Dn:]file-name	Remove the protection caused by the PRO command
@	Begin execution of a batch file
Dn:	Change the default drive number

EXTRINSIC COMMAND SUMMARY

ADOS	Allow access to version 2 and version 4 files at the same time
BASIC	Load and execute BASIC A+
C65	Load and execute the C/65 compiler
CLRDSK	Initialize a diskette like the Atari 810 disk drive does.
CONFIG [parm1 parm2 ...] [-N]	Change the status of a configurable drive
COPY source-file destination-file [-FQSW] or COPY file [-FQSW]	Copy files.
COPY24 source-file destination-file [-FQWD]	Transfer files from version 2 to version 4 diskettes (& vice versa)
DO command[;command;command...] or DO	Perform a sequence of commands
DUPDBL	Duplicate a double density diskette
DUPDSK	Duplicate a diskette
HELP	Provide a MENU of system commands
INIT	Format a diskette
INITDBL	Format a double density diskette on a single drive
MAC65	Load and execute the MAC/65 macro assembler
RS232	Install the serial device handlers ("Rn:") for use with the Atari 850 Interface Module.
SDCOPY source-file destination-file [-FQRV]	Copy single density files to double density diskettes

CHAPTER 1: HOW TO USE THIS MANUAL

Anxious to try OS/A+? Can't wait to wade through all this? No matter what your previous background, we suggest you read chapter 2, which gives an overview of OS/A+ and its commands. At that point, if you are still relatively inexperienced, you will want to read chapter 3, which gives step-by-step instructions for common OS/A+ operations.

If you are a more experienced user, you should read chapters 4, 5, and 6, which cover all the OS/A+ commands in detail and present an explanation of batch processing under OS/A+. Users interested in writing programs for use under OS/A+ will also be interested in chapters 7, 8, and 1Ø, which cover using OS/A+ from BASIC and assembly language.

For users interested in more deeply understanding the internal workings of OS/A+, chapter 9 provides information on the way OS/A+ manipulates disk files.

Even if you are not skipping ahead, you may want to skim through unfamiliar material at first reading; but plan to come back later when you're ready to really understand the system.

Anyway, put your OS/A+ master disk (WITH write protect tab on, PLEASE!) into Drive One , turn on your system, and try us!

P.S. Maybe the first command you want to learn is "DUPDSK" (section 5.1Ø), to back up your valuable system master. Surprised? OS/A+ is NOT copy protected. We hope you are considerate of our rights so that we may continue to be considerate of your convenience.

CHAPTER 2: GETTING STARTED WITH OS/A+

2.1 OVERVIEW OF OS/A+

The purpose of OS/A+ is to provide a way for your Atari computer to communicate with your disk drives, printer, or other computer products. OS/A+ contains commands and utilities which allow you to:

1. Organize information into files on your diskettes.
2. Access this information with ease and precision.
3. Make use of other applications programs (e.g. BASIC A+, MAC/65, BUG/65, Atari BASIC, etc.).
4. Pass control of the computer between the Operating System (OS/A+), Cartridges, and programs stored on disk.

OS/A+ was originally an accident, brought about by the fact that that we developed Atari's DOS and Atari's BASIC on an Apple II computer. To simulate Atari's indeed excellent OS ROMs, we wrote our own simple CIO (Central Input/Output) system. From there, it was only logical that we install a Console Processor similar to that of Digital Research's CP/M (their trademark). Then when we introduced BASIC A+, we moved the "CP" over to the Atari, and presto! There was born OS/A+ version 1.Ø for the Atari.

This manual actually describes two products:
OS/A+ version 2 for Atari Computers
OS/A+ version 4 for Atari Computers

OS/A+ (and, naturally, Atari's OS) utilizes a software concept which is built around a structured and layered scheme. In particular, application programs are expected to make calls to the OS via the Central Input Ouptput routine ("CIO"). In turn, CIO is a dispatcher which examines the application program's request and routes the necessary subrequests to the appropriate device driver(s).

On the Atari, the device drivers may in turn call the SIO (Serial Input/Output) routines to perform the actual channel communications with devices on the serial bus (obvious exceptions include the screen and keyboard, which do not require serial bus service). Finally, the device (on the serial bus) receives the SIO request and performs the actual I/O needed. The diagram on the next page illustrates this process.

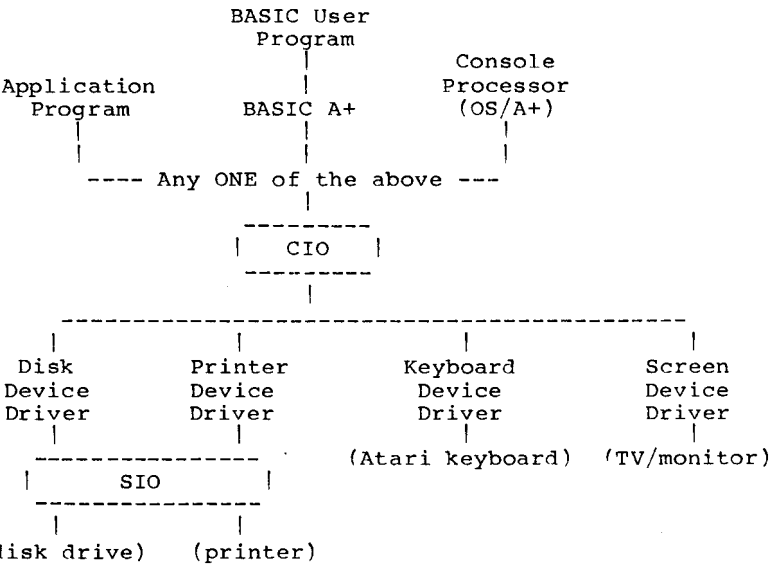

Figure 2-1
Overview of OS/A+

Generally speaking, there is no reason any one or more portions of this hierarchical structure cannot be replaced with another, equivalent section of code. On the Atari computer, in fact, the FMS (or File Management System) itself is "added" to the default structure only if a disk drive is present at power-on time. Several manufacturers, for example, have produced their own printer or screen drivers, replacing the Atari-supplied drivers with minimal effect.

Unfortunately, we cannot say that any given portion may be replaced with NO effect, simply because an unfortunately high portion of software written for the Atari violates the hierarchy (by direct calls to device routines, or worse). These violators are by no means in the majority, or we might have no hope of ever producing an improved Atari system. However, we should be aware of at least the most important of these (quite frankly) poorly written programs and maintain what compatibility that we can when we change the system.

Generally, the worst offenders are programs such as VISICALC and MICROSOFT BASIC, both of which make assumptions about memory layout and disk usage. However, these programs (and most others) are shipped with an operating system intact on the disk on which they reside. Thus, although we may not force them to take advantage of the expanded capabilities that our device drivers may offer, at least we need only maintain compatibility with a standard Atari 800 and 810 Disk Drive to allow their usage on otherwise improved products.

2.2 THE OS/A+ CONSOLE PROCESSOR (CP)

As you might recall from Figure 2-1, the CP (Console Processor) is NOT a priveleged part of the system. CP functions as an easy-to-use interface between the human at the keyboard and the machine level of the CIO calls.

In section 2.1 we mentioned that any portion of the OS/A+ system could be replaced without change to any of the other sections. This is perhaps most true of the CP. For example, in a dedicated run-time environment it has no reason to exist. Others have written an equivalent CP and placed it under the Atari DOS system, but we believe that the CP of OS/A+ is a very well-designed, well-executed human interface, especially considering that it occupies less than 800 bytes of your precious memory.

2.3 SYSTEM REQUIREMENTS

Although both versions of OS/A+ for the Atari will run nicely in 32K bytes of RAM, it isn't realistic to use less than 40K or 48K and expect to do useful work with most languages and/or applications. Obviously, with both versions at least one disk drive is required. Two disk drives are highly recommended. The Atari version 4 system requires (and only runs on) double density or larger disk drives.

2.4 WHY TWO VERSIONS OF OS/A+ ?

Because we like to add to the confusion, of course. Seriously, when we originally produced Atari DOS, we wrote it to Atari specifications. There is more detail on this subject in Chapter 9, but suffice to say the real problem with Atari's FMS (and hence with OS/A+ version 2) is that it was never designed to handle disks larger than 256 Kilobytes. But now Percom Data Company, Software Publishers, and others have added double sided, double-density disks to their catalog, with capacities of nearly 400 Kilobytes.

Given that we need to access more than 256K bytes per disk and/or file, how can we expand on the Atari system? An obvious solution is to introduce the concept of "logical disks", wherein a larger drive might contain two or more disjoint segments, each wholly allocated to imitate an 810 disk system. Anyone who has tried the Corvus equivalent of this scheme will recognize the inadequacies of this solution.

So, given that we will no longer be compatible with Atari products, why not seize this opportunity to "do it right"? Why not produce a wholly different file manager system that is not bound by the restrictions of Atari DOS? This is the path we have chosen.

Thus we come to OS/A+ version 4, a mapped file system. Since we wrote not only Atari DOS but also Apple DOS, we naturally thought of an extension on the Apple scheme as the logical step up from Atari DOS and version 2 of OS/A+. We do not know if it might ever happen, but using our version 4 scheme would, presumably, enable a manufacturer to offer disk systems which were MEDIA and FILE COMPATIBLE on both Apple and Atari (and perhaps other 6502 systems).

2.5 BOOTING UP (and returning to CP)

When an OS/A+ disk is booted, the CP is immediatly entered. Re-entry of OS/A+ from the cartridge is normally done through the use of the DOS command (i.e., the BASIC command for this is DOS). Some cartridges do not allow DOS-type exits and thus OS/A+ cannot be used with these cartridges.

In any case, when the CP is entered it will clear the screen and display:

```
OSS OS/A+   ATARI VERSION x.xx
D1:<cursor>
```

The D1: is the command prompt. It serves two purposes. Firstly, it tells the user it is ready to accept a command. Secondly, it is a reminder of the default disk drive. The default drive, in this case, being the familar file spec for drive 1.

2.6 DISK FILES AND FILENAMES

Most CP commands and parameters deal with files of one sort or another. OS/A+ requires files be specified with a filename of the form:

<device>: <optional-file-name>

The file-specifier may be any valid file name and may contain the "wild-card" characters '?' and '*'. A question mark ('?') will match any character in a file name, while an asterisk ('*') will match any string of zero or more characters. For example, DIR *AB.C?? will match and list XAB.CXX AB.CUR BEOBAB.CNN etc.

The rules for valid file names are:

- Version 2
  - One to Eight characters optionally followed by a period and a one to three character extender.
  - Only characters A-Z and Ø-9 are allowed. Also, the first character must be alphabetic.
- Version 4
  - One to Thirty characters
  - All characters are valid except CTRL characters, RETURNs, and commas (,).

NOTE: under version 4 of OS/A+ the file spec necessary to refer to all files on a disk is just "*", not "*.*"

Device names under OS/A+ are very simplistic; they consist of a single letter optionally followed by a single digit used to define a specific device when more than one of the same kind exist (Ex.- D1: or D2:). Traditionally (and, in the case of Atari disk files, of necessity) the device name is followed by a colon. The following devices are implemented under standard OS/A+:

E: The keyboard/screen editor device. The normal console output.

K: The keyboard alone. Use this device to bypass editing of user input.

S: The screen alone. Can be either characters (ala E:) or graphics.

P: On the Atari, the printer. The standard device driver allows only one printer.

C: The cassette recorder.

D: The disk file manager, which also usually requires a file name.

Other device names are possible (e.g., for RS-232 interfaces), and in fact the ease with which other devices may be added is another mark for the claim that OS/A+ is a TRUE operating system. The structure of device drivers is material for a later section (8.4), but we should like to point out that, on the Atari, the OS ROM includes drivers for all the above except the disk.

To work with the disk file TEST.ORG on disk drive number 1, the operating system requires that the file spec D1:TEST.ORG be used. Having to always specify the D1: can be tedious, especialy if most of the user's file work is on a single drive, CP is designed to prefix all filenames appearing in a CP command line with the default drive - if and only if a device has not been explictly specified. In the case of D1:TEST.ORG, the user could enter only TEST.ORG for a file name and allow CP to prefix it with the default drive. Thus TEST.ORG becomes D1:TEST.ORG in the OS/A+ system. If TEST.ORG happened to be on drive two and the default drive was drive one, the user could enter D2:TEST.ORG; and CP would see that the user has explicitily specified a device and would thus not append the default drive device to that file name.

If the user needs to work a great deal with files on drive two, he can change the default drive to avoid the now necessary D2: prefix typing. When the system prompts D1:<cursor>, the user can respond with D2:<return> to change the default drive to the D2: device. The next CP prompt line will show D2:<cursor>. Now files accessed on drive one will require the explict D1: prefix typing, while files on drive two will not require prefix typing. Only devices of the form Dn: (where n = Ø-7) are allowed as default drives. OS/A+ does not check to insure that the new default drive actually exists. The user's first indication of an invalid default drive will occur when OS/A+ attempts to access a file on the invalid device (via user command). The error message "INVALID DEVICE" will indicate the situation. The user should then set the default device to a valid disk unit. The default device change command is one of the many intrinsic CP commands.

2.7 CP Commands

CP has three general classes, or groups, of commands. The classes are intrinsic commands, extrinsic commands, and execute commands. Intrinsic commands are executed by means of resident code that got loaded into the system when OS/A+ was booted up. Extrinsic commands are commands that are external to the system. That is, the code that is used to execute the command must be loaded into the system from the disk at the time the command is issued. The Execute subset of commands provide for the batch execution of both intrinsic and extrinsic CP commands from a particular file.

The intrinsic commands are explained and defined in Chapter 4, and the extrinsic commands may be found in Chapter 5. Since the execute commands are a subset of the intrinsic they may be found in Chapter 4 too, and will be noted as useful in execute files. They are also discussed in Section 6.3.

CHAPTER 3: HOW TO

Section 3.1

GLOSSARY

Words you need to understand before you begin reading this manual.

Version 2	The name for our operating system that is upward compatible with Atari DOS 2.Øs, with some added features such as the ability to handle double density.
Version 4	The name for our advanced Disk Operating System.
Single density	A standard for storing data on a diskette. Atari 81Ø disk drives are single density disk drives.
Double Density	Twice the standard. A double density diskette will store twice the amount of data stored on a single density diskette.
Double Sided	A disk drive able to use both sides of a diskette to store data.
Single drive	A hardware unit that can hold only one diskette at a time.
Double drive	A hardware unit that can hold two diskettes at one time.
System diskette	A diskette that contains the disk operating system and all, some or none of the utilities that are used with it.
Master diskette	The original diskette that comes with your disk drive. Any diskette duplicated from this diskette would also be called a master diskette.
Boot, Booting	Putting a system diskette in the number 1 disk drive and turning on the system.

{D1:} Anything that is between these braces is meant to be a prompt by the computer. For example, the instruction / Type {D1:}CONFIG 2D / means that with the D1: prompt on the screen you would type in CONFIG 2D.

.* This is the universal filename. When you see this filename in a command line it can be replaced by any and all other filenames. For example, the command line / COPY D1:.* D2: / means that all the files on the diskette in drive 1 will be copyed to the diskette in drive 2.

[RETURN] This symbol represents a carriage return. Whenever it is encountered push the RETURN key on the keyboard.

Section 3.2

BOOTING the MASTER DISKETTE

STEP BY STEP:

1) Configure drive(s) (via switches etc) according to manufacturer's directions.

2) Insert an OS/A+ master diskette in drive 1 and close the door (make sure master diskette has write protect tab in place.)

3) Turn on the drive(s). Wait for the motor to stop.

4) If you will be using a cartridge based systems language (e.g., OSS BASIC or OSS ACTION! or Atari Basic), insert the cartridge in the computer.

5) Turn ON the computer.

6) Wait. Watch and read the screen. (Remember if the screen information is scrolling too fast you may pause by hitting CNTL-1. See Atari's Operator Manual.)

COMMENTARY:

For those of you who own Percom, Software Publishers, Rana or Micro Mainframe disk drives, the switch settings for the master drive (Drive 1, D1:) are not important. When the OS/A+ master diskette is inserted into drive 1 and the machine turned on, the drive will sense the density of the diskette and configure itself properly.

The Atari 810 disk drive operates in single density mode only. When using OS/A+ Version 2 with these drives, step 1 should be omitted.

For all versions of OS/A+, you will notice that as the system is booting some lines of inverse video are being written to the screen. These are from a file called STARTUP.EXC, and is part of the booting operation (more on this file later).

When the system has completed executing the STARTUP.EXC file you will be looking at a menu. At this point you may choose one of the numbers to do a particular command or answer with the number 9 to get control of the system.

In any case, when the D1: prompt appears you have been given control of the operating system, and the boot process will have been completed.

NOTE: the CNTL-1 sequence is executed by pushing down the CTRL key on the left side of the keyboard and at the same time pushing down the number 1 key.

Section 3.3

Booting up directly into a BASIC program

STEP BY STEP:

1) Boot master diskette
2) If you want the startup file on another disk, place that disk in the drive at this time.

3) Type the command:
   {D1:}TYPE E: D1:STARTUP.EXC [RETURN]

The first three steps above are required to create the STARTUP.EXC file. When these steps have been executed the screen will be blanked out and the cursor will be in the top left hand corner of the screen. You are now ready to enter the STARTUP.EXC command line.

4) Type the command:

   DO CAR;RUN"D:MENU" [RETURN]

   Note the filename MENU is a fictitious filename. Please replace this name with a name of a program that is on your disk. Also note that your BASIC program must also have been SAVEd to the disk before it can be used in a STARTUP.EXC file.

5) Type the character:
   (cntl-3)

   To perform the CNTL-3 function, press the key marked CTRL on the left hand side of the keyboard while at the same time pressing down the number 3 key. When step 5 has been executed the file STARTUP.EXC will actually be written to the disk and control will go back to the operating system and the D1: prompt.

6) In answer to the D1: prompt type DIR, to get a directory of the disk. Now if any of the files listed below are not on your diskette, the STARTUP.EXC file will not work properly.

   DOS.SYS
   DO.COM
   STARTUP.EXC
   your BASIC program file that was used in the STARTUP.EXC file

7)

Last but not least, before you try out this newly created diskette by switching the power off and on, make sure the BASIC cartridge is in it's proper slot.

COMMEMTARY

The theory behind the STARTUP.EXC file is to make a diskette boot up and execute a program automaticly. Of course most people will be using this with the BASIC cartridge and BASIC programs as shown in the above example. For those of you with Atari DOS the STARTUP.EXC file operates along the same principles as the AUTORUN.SYS file.

Section 3.4

DUPLICATING A DISKETTE

Version 2, with one disk

STEP BY STEP

1) Boot master diskette
2) For Software Publishers disk drives only:
   Type the command:
   {D1:}CONFIG 1S[RETURN]
3) Type the command:
   {D1:}DUPDSK[RETURN]
   To the following prompts answer as shown:
4) {SOURCE DISK DRIVE (1,2,3,4):} 1[RETURN]
5) {DESTINATION DISK DRIVE (1,2,3,4):} 1[RETURN]
6) {FORMAT DESTINATION DISK (Y OR N):} Y[RETURN]
7) {Put Source Disk in Drive 1
   When Ready, Hit RETURN} [RETURN]
8) Swap source and destination disks as directed by prompts.

COMMENTARY:

Duplicating a diskette with a single disk drive is a little more time consuming than with two disk drives but no less accurate. To begin with, boot the OS/A+ version 2 master diskette. If you are using the Software Publisher disk drive then you must proceed with step 2 since some of these disk drives do not remain in the single density mode when booted.

Once the system is booted and configured properly, the duplicate diskette command can be given (step 3). Remember, the master diskette must still be in drive 1 when the [RETURN] is typed in step 3, otherwise a FILE NOT FOUND error will occur. After the DUPDSK utility has been loaded from the master diskette you'll notice the prompt as in step 4. This prompt is asking you what disk drive your source diskette will be in. Follow step 4's answer and don't worry if your source is not in drive 1 yet; the program will wait for you to insert it before it starts duplicating (step 7).

Since this is a single drive duplication, the destination disk drive will also be drive 1 (Step 5).

The prompt in step 6 is used in case your destination disk is new. We recommend that, whether your disk is new or not, you answer the prompt in step 6 with a Y[RETURN]. Formatting an old diskette that has already been used before will guarantee that all the old files have been completly erased.

Step 7 is where you may take out the master diskette and insert the diskette that you wish to duplicate. After the [RETURN] in step 7, the DUPDSK program takes control and reads in as much of the disk as possible.

When the prompt:

```
Put Destination Disk in Drive 1
 When Ready, Hit RETURN
```

appears it is time to swap disks. At this time take out your source diskette and insert your destination diskette and hit [RETURN]. DUPDSK will format the diskette before writing to it. Then the duplication process will begin.

Note: It might be a good idea to label the diskettes before you start so that the possibility of mixing up your SOURCE and DESTINATION diskettes is greatly reduced.

Version 2 with two disk drives

STEP BY STEP:
---

1) Boot master diskette
2) For Software Publishers disk drives only:
   Type the command:
      {D1:}CONFIG 1S[RETURN]
3) Type the command:
      {D1:}CONFIG 2S[RETURN]
4) Type the command:
      {D1:}DUPDSK[RETURN]
   To the following prompts answer as shown:
5)   {SOURCE DISK DRIVE (1,2,3,4):} 1[RETURN]
6)   {DESTINATION DISK DRIVE (1,2,3,4):} 2[RETURN]
7)   {FORMAT DESTINATION DISK (Y OR N):} Y[RETURN]
8)   {Put Source Disk in Drive 1
       Put Destination Disk in Drive 2
       When Ready, Hit RETURN} [RETURN]

COMMENTARY:

First boot a system diskette. For those of you with Atari disk drives, go to to step 4.

If you have a Software Publishers disk drive you must first configure drive 1 (See step 2).

If you have a non-Atari drive that is software configurable, you must configure drive 2 accordingly (step 3). Remember, version 2 can also handle double density, and some drives support two or more densities. Non-Atari disk drive users should be very careful to make sure both drives are configured properly.

Once the drives are properly configured (if necessary), the duplicate diskette command can be given (step 4). Remember, the master diskette must still be in drive 1 when the [RETURN] is typed in step 4; otherwise a FILE NOT FOUND error will occur.

After the DUPDSK utility has been loaded from the master diskette you'll notice the prompt as in step 5. This prompt is asking you what disk drive your source diskette will be in. Follow step 5's answer and don't worry if your source diskette is not in drive 1 yet, the program will wait for you to insert it before it starts duplicating (step 8).

Since this is a two drive duplication, the destination disk drive will be drive 2 (Step 6).

The prompt in step 7 is used in case your destination disk is new. We recommend that whether your disk is new or not, you answer the prompt in step 7 with a Y[RETURN]. Formatting an old diskette that has already been used before will guarantee that all the old files have been completly erased.

In step 8 you take out the master diskette and insert the diskette that you wish to duplicate. After the [RETURN] in step 8, the DUPDSK program takes control and within a minute or so the diskette in drive 2 will have an exact duplicate of the diskette in drive 1!

Version 4, with one disk drive

STEP BY STEP:
---
1) Boot master diskette
2) For Software Publisher disk drives only:
   Type the command:
      {D1:}CONFIG 2D[RETURN]
3) Type the command:
      {D1:}DUPDSK[RETURN]
   To the following prompts answer:
4)   {SOURCE DISK DRIVE (1,2,3,4):} Type 1[RETURN]
5)   {DESTINATION DISK DRIVE (1,2,3,4):}Type 1[RETURN]
6)   {FORMAT DESTINATION DISK (Y OR N):}Type Y[RETURN]
7)   {Put Source Disk in Drive 1
       When Ready, Hit RETURN} [RETURN]

COMMENTARY:

The process used for duplicating a Version 4 diskette on a single drive, is very similar to the process for duplicating a diskette for a Version 2 single drive. First boot the Version 4 master diskette.

If you have a Software Publishers disk drive you must first configure the drive (See step 2).

With the master diskette still in drive 1, issue the duplicate diskette command (step 3).

After the DUPDSK utility has been loaded from the master diskette you'll notice the prompt as in step 4. This prompt is asking you what disk drive your source diskette will be in. Follow step 4's answer and don't worry if your source diskette is not in drive 1 yet; the program will wait for you to insert it before it starts duplicating (step 7).

Since this is a single drive duplication, the destination disk drive will also be drive 1 (Step 5).

The prompt in step 6 is used in case your destination disk is new. We recommend that whether your disk is new or not, you answer the prompt in step 6 with a Y[RETURN]. Formatting an old diskette that has already been used before will guarantee that all the old files have been completly erased.

In step 7 you take out the master diskette and insert the diskette that you wish to duplicate. After the [RETURN] in step 7, the DUPDSK program takes control and reads in as much of the disk as possible.

When the prompt:

```
Put Destination Disk in Drive 1
 When Ready, Hit RETURN
```

appears it is time to swap diskettes. At this time take out your source diskette and insert your destination diskette and hit [RETURN]. Before DUPDSK starts writing to the diskette, it will format it. Then the duplication process will begin. (Note: It might be a good idea to label the diskettes before you start so that the possibility of mixing up your SOURCE and DESTINATION diskettes is greatly reduced.)

Version 4 with two disk drives

STEP BY STEP
---

1) Boot master diskette
2) For Software Publishers disk drives only:
   Type the command:
        {D1:}CONFIG 1D[RETURN]
3) Type the command:
        {D1:}CONFIG 2D[RETURN]
4) Type the command:
        {D1:}DUPDSK[RETURN]
   To the following prompts answer as shown:
5)   {SOURCE DISK DRIVE (1,2,3,4):} 1[RETURN]
6)   {DESTINATION DISK DRIVE (1,2,3,4):} 2[RETURN]
7)   {FORMAT DESTINATION DISK (Y OR N):} Y[RETURN]
8)   {Put Source Disk in Drive 1
         When Ready, Hit RETURN} [RETURN]

COMMENTARY:
To begin with, the OS/A+ version 4 master diskette must be booted. This process will automaticly configure most disk drives to double density. If you are not sure about this feature of your disk drives then proceed to step 2, otherwise go on to step 3. The reason behind step 2 is that some disk drives (Software Publisher, Inc.) will boot the double density master diskette, but will not leave itself in double density mode, thus you must configure drive 1 to double density.

Step 3 is for everyone, there are no guarantees when booting the system what configuration drive 2 will come up as. Step 3 takes care of any problems, and configures drive 2 to double density.

Once the disk drives have been step up properly (if both drives are not the same density then DUPDSK will not work), the duplicate diskette command is ready to be given (step 4). Remember, the master diskette must still be in drive 1 when the [RETURN] is typed in step 4, otherwise a FILE NOT FOUND error will occur.

Once the DUPDSK utility has been loaded from the master diskette, the prompts in steps 5,6,7 will appear. Answer these prompts as shown above.

Step 8 is where you may take out the master diskette and insert the diskette that you wish to duplicate. After the [RETURN] in step 8, the DUPDSK program takes control and within a minute or so an exact duplicate of the disk in drive 1 will be in drive 2.

Section 3.5

Configuring the drive

This section is for people who have disk drives that are software configurable. If you have only Atari 810 disk drive(s), ignore this section.

The CONFIG command is used to CONFIGure disk drives that are capable of either single or double density (or, in some cases, single or double sided) operations.

Some disk drives are capable of handling a variety of different configurations (e.g. a double sided, double density disk drive can usually ALSO handle single sided, single density diskettes and single sided, double density diskettes). In order to control the various options available, the user must utilize the CONFIG command.

Generally, drive 1 (the boot drive) will automatically be configured to match the density of the boot diskette. CAUTION: Some controllers, such as the Software Publishers unit, do not configure the disk drive upon boot up.

For the Software Publishers disk drives and/or any additional non-Atari disk drive that you may have attached to your system, the CONFIG utility should be used to match the configuration of your disk drive(s) to the configuration of your diskettes.

For example:

> To set up disk drive 2 (D2:), so that it can read single sided, double density diskettes, use the following command (with the master diskette in drive 1).
>
> {D1:}CONFIG 2D [RETURN]

Other examples:

A) {D1:}CONFIG 2S [RETURN] - configure drive 2 to be single side, single density

B) {D1:}CONFIG 1D [RETURN] - configure drive 1 to be single side, double density

C) {D1:}CONFIG 2DD [RETURN] - configure drive 2 to be double side, double density

CAUTION:

If you are using the OS/A+ Version 4 operating system, DO NOT CONFIGure any disk drive to single density. The only exceptions to this rule are detailed under the instructions for ADOS and COPY24.

NOTE:

Whenever the CONFIG command is used a copy of the file CONFIG.COM must be on the diskette in drive 1.

Section 3.6

Formatting a new Version 2 diskette:

STEP by STEP:

1) Boot master diskette
2) Type the command:
   {D1:}INIT [RETURN]
3) To the prompt:
   {1) FORMAT DISK ONLY.}
   {2) FORMAT DISK AND WRITE DOS.SYS.}
   {3) WRITE DOS.SYS ONLY.}
   {4) RETURN TO OS/A+.}

   {ENTER FUNCTION NUMBER: <CURSOR>}
   Type, 2 [RETURN]

4) To the prompt:
   {ENTER DRIVE (1,2,3 OR 4): <CURSOR>}
   Type, 1[RETURN]

5) To the prompt:
   {FUNCTION 2; DRIVE 1}

   {ARE YOU SURE (Y OR N): <CURSOR>}

   Insert a brand new diskette in drive 1 and Type Y[RETURN].

COMMENTARY

The process for formatting a brand new diskette is not as complicated as it looks. The numerous prompts are there to help guide you in creating this new diskette, and not to cause you confussion.

As with all our previous examples, the first step is to boot the OS/A+ Version 2 master diskette. The reason for this is that the proper utility for formatting a new diskette (INIT.COM) will always be on this diskette.

When the master diskette has finished booting and the D1: prompt is on the screen, you are ready to issue the following command (step 2):

   {D1:}INIT [RETURN]

The INIT command will cause the INIT.COM utility to load off the master diskette and issue the following prompt:

1) FORMAT DISK ONLY.
2) FORMAT DISK AND WRITE DOS.SYS.
3) WRITE DOS.SYS ONLY.
4) RETURN TO OS/A+.

These four options of the INIT command allow the user some versatility in creating a new diskette. OPTION 1, will just format the diskette. This disk would more than likely be used to back up programs already on other diskettes. OPTION 2, is used to create a bootable diskette (writting the file DOS.SYS makes the diskette bootable). OPTION 3 would only be used if the diskette being inserted into the drive is already formatted. And of course option 4 allows a quick exit in case you change you mind.

We will choose option 2, to create a brand new bootable diskette. So, to the prompt:

ENTER FUNCTION NUMBER:

enter the number 2 and hit [RETURN].

Now that the system knows what you want to do (format diskette and write DOS.SYS) it has to know where you want to do it (what disk drive). This question gets answered with the following prompt:

ENTER DRIVE (1,2,3 OR 4):

Drive 1 is where this process will take place, so enter 1 [RETURN].

The computer now has all the information it needs to know for creating the new diskette, but before it starts, it will issue the following prompt to confirm your choices and allow you to insert your new diskette.

FUNCTION 2; DRIVE 1

ARE YOU SURE (Y OR N):

If the function and drive number are not correct then type N[RETURN]. If the function and drive number are correct then remove the OS/A+ master diskette and insert your brand new diskette in drive 1. Now answer the question with a Y[RETURN] and within a minute or so a brand new bootable diskette will be in drive 1.

NOTE: Once a diskette has been formatted the user may copy any file to this diskette.

Section 3.7

Formatting a new Version 4 diskette:

STEP by STEP:
---

1) boot OS/A+ Version 4 master diskette.

2) Type the command:
   {D1:}INIT [RETURN]

3) To the prompt:
   DRIVE NUMBER ?

   Type, 1 [RETURN]

4) To the prompt :
   INSERT DISK INTO DRIVE 1
   AND HIT RETURN WHEN READY

   Remove your master diskette and insert a brand new double density diskette and type 1[RETURN].

5) To the prompt:
   INITIALIZATION COMPLETE
   INIT ANOTHER DISK?

   Type N[RETURN]. This causes control to pass back to the operating system

6) Remove newly formatted diskette and reinsert OS/A+ master diskette.

7) Type the command:
   {D1:}COPY D1:DOS.SYS D1: -SW[RETURN]

8) To the following prompt:
   Insert disk(s) to be copied
   and hit return when ready
   just type [RETURN]

9) To the following prompt:
   Insert 'to' disk and hit return

Put your newly formatted diskette in drive 1 and type [RETURN]

COMMENTARY:

As with all our previous examples, the first step is to boot the OS/A+ Version 4 master diskette. The reason for this is that the proper utility for formatting a new diskette (INIT.COM) will always be on this diskette.

When the master diskette has finished booting and the D1: prompt is on the screen, you are ready to issue the following command:

{D1:}INIT [RETURN]

This command will cause the INIT.COM utility to load off the master diskette and issue the following prompt:

DRIVE NUMBER ?

This prompt is asking, in what disk drive will the format process be performed. Answer this prompt with a 1[RETURN], for disk drive number 1.

Now that the system knows what it is doing (formatting a new double density diskette), and where it will be doing it (disk drive number 1), the next prompt will be issued.

INSERT DISK IN DRIVE 1
AND HIT RETURN WHEN READY

It is now time to remove your OS/A+ master diskette and insert you brand new double density disk. When you hit [RETURN] the system will start formatting the diskette in drive 1, so be sure to remove the master first before hitting [RETURN]

When the initialization is complete the system issues the next prompt.

INITIALIZATION COMPLETE
INIT ANOTHER DISK?

Type N[RETURN] to get control of the operating system. With the D1: prompt on the screen, remove your newly formatted diskette and place it aside for a moment. Now reinsert your OS/A+ master diskette and type the following command:

{D1:}COPY D1:DOS.SYS D1: -SW [RETURN]

Even though you have formatted this new diskette you have set aside, it will not boot until the file DOS.SYS has been copyed to it. Executing the command above causes the COPY.COM utility on the master diskette to load in to the system, and issue the following prompt:

```
Insert disk(s) to be copied
and hit return when ready
```

Since the file DOS.SYS is already on your master diskette, there is no need to insert another source diskette. So with the master diskette still in drive 1 answer the prompt above with a [RETURN].

Answering the above prompt with a [RETURN] cause the file DOS.SYS located on the master diskette to be read into memory. When the file DOS.SYS have been completly read into memory the prompt below will be issued.

```
Insert 'to' disk and hit return
```

When this prompt appears on the screen, remove your master diskette from drive 1 and insert your newly formatted double density diskette you set aside. Once the destination diskette has been inserted type a [RETURN]. This will cause the file DOS.SYS to be written to this new diskette. With this complete the diskette in drive 1 in now bootable.

Section 3.8

Copy files with only one drive

STEP BY STEP:

1) Boot master diskette
2) Type the command:
   {D1:}COPY D1:**.* D1: -SWQ [RETURN]
3) To the following prompt:
   Insert disk(s) to be copied
   and hit return when ready
   insert your source disk in drive 1 and hit [RETURN]
4) Follow the prompts answering either Y or N with a [RETURN].

COMMENTARY:

The COPY command is used to copy a single file or mutiple files from one disk to another. But, before you can do any copying you must put a disk in drive 1 that has a copy of the utility COPY.COM, whether the files you want to copy are on that disk or not. This is done by using the master diskette.

Now, with the master diskette in drive 1, the command line (step 2) can be issued to the operating system. This command line,

{D1:}COPY D1:**.* D1: -SWQ [RETURN]

tells the computer to COPY all the files on the disk in drive 1 (source disk) to another disk (destination disk) that you will be subsituting with the source disk.

The letters at the end of the command line, called flags, give the copy utility some information about this particular copy. The -S flag tells the COPY utility that this is a single drive COPY. If you have only one disk drive the -S flag must always be used.

The -W flag, tells the system to Wait before it actually starts to perform the copy. Remember, that in step 1 the master diskette is in drive 1. If the -W flag is not used in step 2 the system will load the COPY utility off the master diskette and assume that the master diskette is also your source diskette for COPYing. When the -W flag is used the COPY utility is loaded from the master diskette and the system will Wait for you to insert the proper source diskette.

The -Q flag when used, forces the system to Query (question) the user as to whether the file should be COPYed or not. A prompt, like the one below will ask the user about COPYing the file. This prompt can be answered with either a Y[RETURN] or N[RETURN].

```
COPY D1:filename
  TO D1:filename?
```

If you answer Y[RETURN] then that particular file will be read into the computers memory. Once the file has been read into memory, the computer will prompt you to subsitute your destination disk for the source disk. After you have completed the subsitution and hit [RETURN], the file will now be COPYed to you destination disk.

If you type N[RETURN] then the system will respond:

```
D1:filename NOT COPIED
```

This is just to confirm the fact that this particular filename was not copied to the destination disk.

For single file copies with 1 drives use the command line:

```
COPY D1:source-filename D2:
```

this example expects to see the file COPY.COM as well as your source filename on the same disk. If your source filename is on another disk, then you must use the -W option so that after the COPY utility is read in it will Wait for you to swap disks.

Single drive users please DON'T:

1) Copy the file DOS.SYS without first renaming it. (OS/A+ Version 2 users only)

2) Use wild card characters in the destination filename

3) Use the COPY command without first having a copy of COPY.COM on your disk

COPY with 2 drives

STEP BY STEP:

Configurable drive users with OS/A+ Ver 4

1) Boot master diskette
2) Type the command:
   {D1:}CONFIG 2D [RETURN]
3) Type the command:
   {D1:}COPY D1:**.* D2: -WQ [RETURN]

Configurable drive users with OS/A+ Ver 2

1) Boot master diskette
2) Type the command:
   {D1:}CONFIG 2S [RETURN]
3) Type the command:
   {D1:}COPY D1:**.* D2: -WQ [RETURN]

Atari 810 users with OS/A+ Ver 2

1) Boot master diskette
2) Type the command:
   {D1:}COPY D1:**.* D2: -WQ [RETURN]

COMMENTARY:

You will notice that in all three examples above, the first step is to boot the master diskette. The reason for this is that the utility COPY.COM will always be on this diskette, and before we do any COPYing we must first load the COPY.COM utility. So whether you are using Atari disk drives or Percoms please boot the master diskette before any COPYing is performed.

In our Percom examples above, you will notice that Step 2 uses the CONFIG command. This command is used to make sure that your destination disk drive (D2:) is configured the same as your source disk drive (D1:). In fact after the CONFIG command is executed a chart will be printed on the screen so that you can see that both disk drives have the same configuration. In the example with the Atari disk drive there is no CONFIG command because the Atari disk is non-configurable. Because of this you cannot use the 810 disk drive and the COPY command with OS/A+ Version 4.

Once the disk drive has been configured properly the COPY command is ready to be executed. The COPY command line in all these examples, will copy all the files on your diskette in drive 1 to the diskette in drive 2.

The letters at the end of the command line, called flags, give the copy utility some information about this particular copy. The -W flag, tells the system to Wait before it actually starts to perform the copy. Remember, that in step 1 the master diskette is in drive 1. If the -W flag is not used in step 3 the system will load the COPY utility off the master

diskette and assume that the master diskette is also your source diskette for COPYing. When the -W flag is used the COPY utility is loaded from the master diskette and the system will Wait for you to insert the proper source diskette.

The -Q flag when used, forces the system to Query (question) the user as to whether the file should be COPYed or not. A prompt, like the one below will ask the user about COPYing the file. This prompt can be answered with either a Y[RETURN] or N[RETURN].

```
COPY D1:filename
  TO D1:filename?
```

If you answer Y[RETURN] then that particular file will be COPYed to the diskette in drive 2.

If you type N[RETURN] then the system will respond:

```
D1:filename NOT COPIED
```

This is just to confirm the fact that this particular filename was not copied to the destination diskette.

Section 3.9

Use of COPY24 with one disk drive

STEP BY STEP:

1) Boot OS/A+ Version 4 master diskette
2) Type the command:
   {D1:}ADOS [RETURN]
3) Type the command:
   {D1:}COPY24 A1:**.* D1: -Q [RETURN]
   To the following prompt:
   Insert disk(s) to be copied
   and hit return when ready

4) Insert Atari DOS or OS/A+ Version 2 diskette

COMMENTARY:

This utility is for users with OS/A+ Version 4 only. If you are using another version of the operating system please skip this section, as it does not apply to you.

The COPY24 utility is for file conversion. Since OS/A+ Version 4 has a file structure different from Atari DOS 2.ØS and OS/A+ Version 2, files created under the later two 2 systems must be converted before they will work under the Version 4 system.

To convert either Atari DOS or OS/A+ Version 2 files to the OS/A+ Version 4 system, begin with booting the master diskette. Doing this eliminates the need to go searching for the particular utilities that are needed for the conversion.

To do this conversion the utility ADOS must first be loaded into the system (step 2). ADOS is actually the file manager that is part of the Atari DOS and OS/A+ Version 2 operating system and will be used to read files from those diskettes so that they can be converted.

Once ADOS has been loaded the actual command line for the conversion can be issued to the system (step 3). This command line,

{D1:}COPY24 A1:**.* D1: -Q [RETURN]

tells the computer that ALL the files on either your Atari DOS diskette or OS/A+ Version 2 diskette are to be converted and written to your OS/A+ Version 4 diskette. Also notice, that the (A1:**.*) and not D1:, is used to specify the Atari DOS or OS/A+ Version 2 files. Using this convention also tells the system that it must first reconfigure the disk drive to single density before it can read anything from the diskette known as A1:

After the COPY24 utility has been load the following prompt will be issued.

```
Insert disk(s) to be copied
and hit return when ready
```

At this time you will insert your Atari DOS or OS/A+ Version 2 diskette in drive 1. When you type the [RETURN] the system will reconfigure itself to single density and begin to read the Atari DOS or OS/A+ single density diskette.

The letter at the end of the command line, called a flag, give the COPY24 utility some information about this particular conversion.

The -Q flag when used, forces the system to Query (question) the user as to whether the file should be COPYed or not. A prompt, like the one below will ask the user about COPYing the file. This prompt can be answered with either a Y[RETURN] or N[RETURN].

```
COPY A1:filename
  TO D1:filename?
```

If you answer Y[RETURN] then that particular file will be read into the computers memory. Once the file has been read into memory, the computer will prompt you to subsitute your double density OS/A+ Version 4 destination diskette for the source diskette. After you have completed the subsitution and hit [RETURN], the file will now be converted and COPYed to your destination diskette.

If you type N[RETURN] then the system will respond:

```
A1:filename NOT COPIED
```

This is just to confirm the fact that this particular filename was not copied to the destination diskette.

For single file copies with 1 drive use the following command line after the ADOS utility has been loaded:

```
COPY24 A1:source D1:destination
```

This form (where "source" must be the name of a single density Atari DOS or OS/A+ Version 2 file and "destination must be the name of your double density OS/A+ Version 4 diskette) expects to see the file COPY24.COM on your OS/A+ Version 4 diskette that is in drive 1.

Single drive users please DON'T:

1) Copy the file DOS.SYS with COPY24
2) Use wild card characters in the destination filename
3) Use the COPY24 command without first having a copy of COPY24.COM on your diskette
4) Use the COPY24 utility without first loading the ADOS utility.
5) Use the CONFIG command for single drive copying. COPY24 performs the configuration to single density immediately before copying the file.

USE of COPY24 with 2 drives

STEP BY STEP:
1) Boot OS/A+ Version 4 master diskette
2) Type the command:
   {D1:}CONFIG 2S[RETURN]
3) Type the command:
   {D1:}ADOS[RETURN]
4) Type the command:
   {D1:}COPY24 A2:**.* D1: -WQ[RETURN]

COMMENTARY:

This utility is for users with OS/A+ Version 4 only. If you are using another version of the operating system please skip this section, as it does not apply to you.

As is the case with all our examples, the first step is to boot the master diskette. The reason for using this diskette is that all the utilities provided with the operating system are on this diskette.

If we are going to copy a single density file to a double density file, one of the disk drives must be configured single density. This is done is step 2 with the CONFIG command. After this command has been executed, you will notice that the chart printed on the screen looks like this:

drive	no. sides	density
1	1	double
2	1	single
.	.	.

This means that drive 1 will be expecting to see a single sided double density diskette, while drive 2 will be expecting to see a single sided, single density diskette. In other words you will be putting your OS/A+ Version 4 diskette (destination diskette) in drive 1 and your OS/A+ Version 2 or Atari DOS 2.Øs diskette (source diskette) in drive 2.

To do this conversion the utility ADOS must first be loaded into the system (step 2). ADOS is actually the file manager that is part of the Atari DOS and OS/A+ Version 2 operating system and will be used to read files from those diskettes so they can be converted.

Once ADOS has been loaded the actual command line for the conversion can be issued to the system (step 3). This command line,

```
COPY24 A2:**.* D1: -WQ[RETURN]
```

tells the computer that ALL the files on either your Atari DOS diskette or OS/A+ Version 2 diskette are to be converted and written to your OS/A+ Version 4 diskette. Also notice, that the (A2:**.*) and not D2:, is used to specify the Atari DOS or OS/A+ Version 2 files. Using this convention also tells the system that the diskette known as A2: is single density.

After the COPY24 utility has been load the following prompt will be issued.

```
Insert disk(s) to be copied
and hit return when ready
```

At this time you would insert your Atari DOS or OS/A+ Version 2 diskette in drive 2 and your OS/A+ Version 4 diskette in drive 1. (Note: we would suggest that a newly initialized OS/A+ Version 4 diskette be placed in drive 1 at this time.)

The letters at the end of the command line, called flags, give the copy24 utility some information about this particular copy. The -W flag, tells the system to Wait before it actually starts to perform the coversion. Remember, that in step 1 the master diskette is in drive 1. If the -W flag is not used in step 4 the system will load the COPY24 utility off the master diskette and assume that the master diskette is also your destination diskette for the converted files When the -W flag is used the COPY24 utility is loaded from the master diskette and the system will Wait for you to insert the proper source and destination diskettes.

The -Q flag when used, forces the system to Query (question) the user as to whether the file should be converted or not. A prompt, like the one below will ask the user about converting the file. This prompt can be answered with either a Y[RETURN] or N[RETURN].

```
COPY D1:filename
  TO D1:filename?
```

If you answer Y[RETURN] then that particular file will be converted and written to the OS/A+ Version 4 diskette, in drive 1.

If you type N[RETURN] then the system will say:

A1:FILENAME NOT COPIED

This is just to confirm the fact that this particular filename was not copied to the destination disk.

Please DON'T:

1) Copy the file DOS.SYS with COPY24
2) Use wild card characters in the destination filename
3) Use the COPY24 command without first having a copy of COPY24.COM on your diskette
4) Use the COPY24 utility without first loading the ADOS utility.

Section 3.10

Single density to Double density copy (SDCOPY)
with 1 drive

STEP BY STEP
1) Boot OS/A+ Version 2 master diskette
2) Type the command
   {D1:}INITDBL[RETURN]
3) Remove master diskette and insert a blank disk
   into drive 1.
4) To the following prompts answer as shown:
   DRIVE TO INITIALIZE? 1[RETURN]
   INSERT DISK AND HIT RETURN [RETURN]
5) Reinsert master diskette
6) Type the command:
   {D1:}SDCOPY D1:**.* D1: -WQ [RETURN]
7) To the following prompt answer as shown:
   Insert disk(s) to be copied
   and hit return when ready[RETURN]

COMMENTARY

This command is for OS/A+ Version 2.1 users only. If you are not using this version of the operating system please skip this section.

The utility SDCOPY is used to make double density copies of your single density files. However, before SDCOPY can be used correctly, the user must first set up properly.

To begin with, boot the OS/A+ Version 2.1 diskette. Doing this eliminates the need to go searching for the diskette that has the proper utilities on it.

With the master diskette in drive 1 use the INITDBL command to create a double density formatted diskette (steps 2-6). When the prompt in step 4 appears insert a brand new double density diskette, then hit [RETURN].

When the disk drive motor turns off, take out your newly formatted double density disk and lay it to the side. This diskette will be your destination diskette when using the SDCOPY utility.

Reinsert your OS/A+ Version 2.1 master diskette, and type the command line in step 6. This command line,

{D1:}SDCOPY D1:**.* D1: -WQ [RETURN]

tells the computer to COPY all the files on the disk in drive 1 (source diskette) to your double density diskette (destination diskette) that you will be subsituting with the source diskette.

The letters at the end of the command line, called flags, give the SDCOPY utility some information about this particular copy. The -W flag, tells the system to Wait before it actually starts to perform the copy. Remember, that in step 5 the master diskette is in drive 1. If the -W flag is not used in step 6 the system will load the SDCOPY utility off the master diskette and assume that the master diskette is also your source diskette for COPYing. When the -W flag is used the SDCOPY utility is loaded from the master diskette and the system will Wait for you to insert the proper source diskette.

The -Q flag when used, forces the system to Query (question) the user as to whether the file should be COPYed or not. A prompt, like the one below will ask the user about COPYing the file. This prompt can be answered with either a Y[RETURN] or N[RETURN].

```
COPY D1:filename
  TO D1:filename?
```

If you answer Y[RETURN] then that particular file will be read into the computers memory. Once the file has been read into memory, the computer will prompt you to subsitute your double density destination diskette for the source diskette. After you have completed the subsitution and hit [RETURN], the file will now be COPYed to you destination diskette.

If you type N[RETURN] then the system will say:

```
D1:FILENAME NOT COPIED
```

This is just to confirm the fact that this particular filename was not copied to the destination diskette.

For single file SDCOPies use the command line:

```
SDCOPY D1:source filename D1:
```

this example expects to see the file SDCOPY.COM as well as your source filename on the same diskette. If your source filename is on another diskette, then you must use the -W option so that after the SDCOPY utility is read in it will Wait for you to swap diskettes.

Single drive users please DON'T:

1) SDCOPY the file DOS.SYS without first renaming it.
2) Use wild card characters in the destination filename.
3) Use the SDCOPY command without first having a copy of SDCOPY.COM on your diskette.

CHAPTER 4 - INTRINSIC OS/A+ COMMANDS

The intrinsic commands described in this chapter are executed via code that was loaded into the system at bootup time. These commands do not require the loading of programs to perform their functions (as do extrinsic commands). The following is a summary of the most often used intrinsic commands:

```
DIRECTORY - List Directory
PROTECT   - Protect a file (from change or erase)
UNPROTECT - Unprotect a file
ERASE     - Erase (delete) a file
RENAME    - Renames a file
LOAD      - Load a binary file
SAVE      - Save a binary file
RUN       - Execute a program at some address
CARTRIDGE - Run Atari cartridge in the "A"
            cartridge slot (Atari users only)
TYPE      - Type a text file to the screen
```

(The default drive change command, Dn:, is also considered an intrinsic command.)

All intrinsic commands may be abbreviated to their fi three characters. As a matter of fact, OS/A+ only looks the first three characters while testing for an intrins command. Each of the commands will be covered in det later in this manual; however, to give you a feel of intrinsic commands, let's look at the DIRECTORY comma While looking at these examples, assume the D1: is default device and has been placed on the screen by CP.

```
D1:DIRECTORY   list all files of disk on drive one
D1:DIRECT        "   "     "   "   "     "   "    "
D1:DIRTY         "   "     "   "   "     "   "    "
D1:DIR           "   "     "   "   "     "   "    "
D1:DIR **.*      "   "     "   "   "     "   "    "
D1:DIR D1:       "   "     "   "   "     "   "    "
D1:DIR D1:**.*   "   "     "   "   "     "   "    "
D1 DIR D2:     list all files of disk on drive two
D1:DIR D2:**.*   "   "     "   "   "     "   "    "
D1:DIR *.OBJ   files with extension .OBJ on drive one
D1:DIR D2:*.ASM files with extension .ASM on drive two
```

Section 4.Ø
---

command: @

purpose: This command begins execution of a batch command file

usage: @file-name

arguments: The name of a .EXC file containing CP commands. The name should be used WITHOUT the .EXC extension.

Description

The @ command tells OS/A+ to begin taking commands from a batch file. This file is a text file which may contain both intrinsic and extrinsic OS/A+ commands. For example, suppose the file TEST.EXC contains the following commands:

```
DIR  D:
DIR  D2:
END
```

Issuing the command

```
@TEST
```

would tell OS/A+ to start taking commands from the file TEST.EXC. At that point, a directory listing of drive 1 would be given, followed by a listing of files on drive 2.

See sections 3.3 and 6.1 for more information on creating and using batch files.

NOTE: The .EXC extension should NOT be given as part of the file-name when issuing the @ command. The command @GEORGE is sufficient to begin execution of the file GEORGE.EXC. In fact, an error will result if the command @GEORGE.EXC is tried.

NOTE: A CAR command, when encountered within a batch file will stop batch execution.

Section 4.1
---

command: CAR

purpose: This command transfers control to a cartridge

users: Atari users only

usage: CAR

arguments: none

Description

The CAR command allows the user to enter a cartridge from OS/A+. The cartridge will retain control of the system until a DOS command is executed from the cartridge.

CAUTION: Some cartridges do not allow DOS-type exits and thus OS/A+ cannot be used with these cartridges.

WARNING: If no cartridge is present, using this command may cause the keyboard to lock up, rendering the machine useless. To rectify this condition, turn off the computer power and reboot the system.

Section 4.2

command: DIRectory

purpose: The command allows the user to view the disk directory

usage: DIR [Dn:][file-name] [output file-spec]

arguments: optional file specifier
optional output file specifier

Description

The DIR command searches the disk directory of the specified disk (or the current default drive, if Dn: is omitted) for all files matching the file-specifier. The names of all files matching the specifier are then printed to the screen, together with the length of the file (in sectors). An asterisk preceding the file's name indicates that the file is protected from erasure, writing, or renaming.

The file-specifier may be any valid file name (see sections on file structure) and may contain the "wild-card" characters '?' and '*'. A question mark ('?') will match any character in a file name, while an asterisk ('*') will match any string of zero or more characters. For example,

```
DIR *AB.C??
```

will match and list

```
XAB.CXX
AB.CUR
BEOBAB.CNN
  etc.
```

If the output file name is specified, the directory listing will be sent to that file instead of to the screen. For example, the command

```
DIR D1: P:
```

will send to the printer a listing of all files on drive 1.

Section 4.3

command: END

purpose: Stop batch execution from within an execute file

usage: END

arguments: none

Description

The END command causes OS/A+ to stop reading commands from a batch file and to resume prompting the user for commmands. This command has no effect outside of a batch file.

Section 4.4

command: ERAse

purpose: This command removes files from a disk

usage: ERA [Dn:]file-name

arguments: a file specifier string

Description

The ERA command permanently removes files from a disk. All files matching the file-specifier string on the specified drive (or the current default drive, if Dn: is omitted) will be erased from the disk. These files will no longer be shown when a DIR command is issued, nor will they be available for any type of file access.

WARNING: As this command causes the irreversible deletion of files from the disk, it should be used with care. Use the PROtect command to guard files against accidental erasure.

Examples:

ERASE *.BAK
will erase all files with an extension of .BAK that are unprotected and that reside on the current default drive.

ERA D2:DUP.SYS
will erase the file named DUP.SYS from disk in disk drive number 2.

Notes:

If ERAse does not find any erasable files that match the specifier, it will return a FILE NOT FOUND error.

Section 4.5

command: LOAd

purpose: Load disk files into memory

usage: LOAD [Dn:]file-name

arguments: a file specifer

Description

The LOAD command allows the user to load binary load image files into user memory. The files must be compatible with the normal binary object files used by the normal host computer operating system. That is:

For Atari users, each segment of the memory image file must be preceeded by two addresses, the starting and ending addresses in RAM memory of the segment. The entire file must be preceeded by two bytes with all bits on ($FF, $FF). This format is identical to that produced by Atari's Assembler/Editor Cartridge and most upgraded products (including ACTION and MAC/65 from OSS).

Section 4.6

command: NOScreen

purpose: Turns off command echo to screen during batch

usage: NOS

arguments: none

Description

Normally, all commands encountered during batch execution are echoed to the screen as if they were typed in by the user. The NOS command can be used to prevent this echo. All commands within an execute file will then no longer be echoed until the execute file is stopped for any reason or a SCR command is encountered.

This command only effects commands encountered in batch mode.

Section 4.7

command: PROtect

purpose: This command protects files from accidental erasure, writing, or renaming

usage: PRO [Dn:]file-name

arguments: a file specifier

Description

The PRO command allows the user to protect one or more files from any erasure, writing, or renaming. All files matching the file-specifier will be protected. The system marks a protected file by placing an asterisk next to its name whenever a DIR command is used. The UNP command can be used to disable the protection, when desired.

Section 4.8

command: REMark

purpose: Prints remarks to the screen during batch execution

usage: REM any characters

arguments: a string of zero or more characters

Description

The REM command performs no operation whatsoever. Its sole purpose is to provide a means of easily printing messages to the screen from an executing batch file (see section on batch execution). When encountered during batch execution, the command line containing the REM command will be echoed to the screen, unless the NOScreen command has been previously issued.

Section 4.9

command: RENname

purpose: Rename a file to a new name

usage: REN from-file-name to-file-name

arguments: two file names

Description

The REN command will search the specified disk (or the default drive, if Dn: is not specified) for a file whose name matches the specified from-file-name. If the file is found, its name will be changed to the indicated to-file-name. An error occurs if the from-file is not found on the disk. The to file-name should NOT be preceeded by a disk drive specifier.

WARNING: The RENname command should not be used with wild-card characters ("*","?") in the file names. Such usage may permanently damage your diskette directory.

WARNING: Under version 2 and Atari Dos 2.ØS, it is possible to use the rename command to create two files with the same name. If this condition occurs, use the COPY command with the query (-Q) option to transfer the two files to separate disks where they may then be renamed back.

Section 4.10

command: RUN

purpose: This command transfers control to an address in memory

usage: RUN [hex-address]

arguments: an optional hexadecimal address

Description

The RUN command immediately causes OS/A+ to perform a jump to the indicated address (or to the address contained in the OS/A+ RUNLOC, if no address is given). The hex-address, if present, must consist of 3 or 4 hexadecimal digits.

The address in RUNLOC is set any time an extrinsic command is issued or a program is loaded using the LOAD command. Therefore, the RUN command may be used to reenter a program such as BASIC after leaving the program through a DOS command.

IMPORTANT NOTE:

Most standard OS/A+ interactive system programs will set RUNLOC to point to their warmstart entry point. Thus, for example, if the user returns to DOS in order to perform an INTRINSIC command, he/she may reenter the systems program by simply typing RUN. At the current writing, BASIC A+ and MAC/65 (for example) both follow this protocol: simply type RUN from CP to reenter at their warmstart points.

Section 4.11
---

command: SAVe

purpose: Save a portion of memory to a disk file

usage: SAVE file-spec start-address end-address

arguments: a file specifier
a hexadecimal starting address
a hexadecimal ending address

Description

The SAVE command allows the user to write portions of memory to disk files in standard binary file format. The two addresses define the portion of memory to be written to disk; the second address must be greater than or equal to the first. A file which has been 'saved' may be later returned to memory using the LOAD command.

Example:

At the time of this writing, the BASIC A+ user with an Atari computer having 48K Bytes of RAM could patch the distribution copy of BASIC A+ and save the new patched version to disk via

SAVE NEWBASIC.COM 8400 BC00

(PLEASE verify these addresses in your BASIC A+ manual; they ARE subject to change.)

Section 4.12

command: SCReen

purpose: Cause batch commands to be echoed to the screen

usage: SCR

arguments: none

Description

The SCR command causes commands encountered during batch execution to be echoed to the screen. The NOS command may be used to turn off the echo of batch commands.

This command only effects commands encountered in batch mode.

Section 4.13

command: TYPe

purpose: This command types an ascii or atascii file to the screen or another file

usage: TYP [Dn:]file-name [output-file]

arguments: filename - the name of any text file.
output-file an optional output file.

Description

The TYPe command allows the user to copy text files to the screen or another file. If the optional output file is not specified, the text file indicated will be copied to the screen. For example, to view the commands in the STARTUP.EXC file on your OSS master diskette, issue the command

```
TYP STARTUP.EXC
```

If the optional output file is specified, the text file will be copied to the output file. For example, to copy the STARTUP.EXC file to the printer, issue the command

```
TYP STARTUP.EXC P:
```

The TYPe command may also be used to copy text files from one disk file to another by using a disk file name as the output file.

Section 4.14

command: UNProtect

purpose: This command removes the protection caused by the PRO command

usage: UNP [Dn:]file-name

arguments: a file specifier

Description

The UNP command allows the user to remove the write protection caused by the PRO command so that files may again be erased, renamed, or written to. All files matching the file-specifier on the specified drive (or the current default drive, if Dn: is omitted) will be affected. These files will no longer be shown with a preceding asterisk when the DIR command is used.

CHAPTER 5: EXTRINSIC OS/A+ COMMANDS

The extrinsic commands are programs that are run by OS/A+. Any binary file containing the .COM extension may be used as an OS/A+ extrinsic command. The OS/A+ COPY command is one such extrinsic command. If you perform the OS/A+ DIRECTORY command on the master diskette, you will see a file named COPY.COM. The program in the COPY.COM file is what is executed when the COPY command is typed.

Remember, extrinsic commands are external to the operating system. Whenever an extrinsic command is executed from OS/A+, the system MUST go looking on the diskette for a .COM file associated with the particular extrinsic command issued and load that file into the system. For example, when the extrinsic command DUPDSK is executed the system will go looking on the diskette in drive 1 for a file call DUPDSK.COM. If the command's .COM file is not on the diskette the system will return a FILE NOT FOUND error. So remember: whenever you issue an extrinsic command to the system its .COM file must be on the diskette for the command to execute properly.

Whenever the user types a command to OS/A+, the command (first three characters only) is compared to the intrinsic command list. If the command is not in the intrinsic list, it is assumed to be extrinsic. A consequence of this is that no extrinsic command program may start with three characters which match any of the intrinsic commands. For example, a program named "PROCESS3.COM" could not be call by simply typing "PROCESS3", since OS/A+ would view that as the intrinsic command "PROtect". Solutions:

(1) Rename the extrinsic command file.
(2) Type the commands:

```
LOAD PROCESS3.COM [RETURN]
RUN [RETURN]
```

To process an extrinsic command, OS/A+ will:

1) Prefix the command with the default device (if a device is not specified).
2) Attach the .COM extension to the command.
3) Open the generated file spec for input.
4) Test file for program of proper LOAD file format.
5) Load and execute the program.

NOTE: (i) If any element of the procedure fails, various error messages will result.
(ii) Step 1 of the procedure implies that a device may be specified. This is in fact the case.

Never explicitly specify the .COM extension as part of the command. The command COPY.COM will result in a file spec of D1:COPY.COM.COM, which is invalid.

Some extrinsic commands (such as COPY) are supplied by OSS. The number of possible extrinsic commands is not, however, limited to these few; commands may be written by the user to perform virtually any function. If you are intrested in writing your own extrinsic commands, see Chapter 8.

If an extrinsic command (i.e., a program running in RAM) has control, the program may generally be rerun or reentered by simply using the RUN command without parameters. Execptions to this rule are the extrinsic commands COPY, COPY24, SDCOPY and CONFIG.

This chapter gives a description of each extrinsic command supplied as a standard part of an OS/A+ system master diskette (except that some commands may be specific to particular versions or packages).

Section 5.1

command: ADOS

purpose: This command allows access to version 2 and version 4 files at the same time

users: Atari version 4 users only

usage: ADOS

arguments: none

options: none

Description

This program installs OS/A+ version 2 along with version 4. This allows the user to access version 4 disks as Dn: while accessing version 2 disks as An:. The usage of this command does require more memory for the DOS, so the low memory pointer (LOMEM, $2E7) will be moved up accordingly. In order to restore the system to its former state (i.e., version 4 only), press the system reset key.

After using ADOS to install the version 2 file system, you may use COPY24 to copy version 2 files to version 4 diskettes, or vice versa.

WARNING: The DUPDSK and INIT commands must not be used while the ADOS command is in effect! This will result in a system crash.

WARNING: If the CONFIG command is issued while ADOS is installed, the ADOS command must again be issued.

Section 5.2

command: BASIC

purpose: This command loads and executes the BASIC A+ language

users: BASIC A+ disk owners only

usage: BASIC [Dn:][file-name]

arguments: optionally, the name of a saved BASIC A+ program

options: none

Description

This command loads and executes the file BASIC.COM, which is the BASIC A+ language. If the optional file name is specified, BASIC A+ will automatically load and execute that file.

NOTE: The file must have previously been 'SAVE'd from BASIC A+. Refer to the BASIC A+ manual for information specific to the language.

*** FOR MORE INFORMATION, SEE YOUR BASIC A+ MANUAL ***

Section 5.3

command: C65

purpose: this command loads and executes the OSS C/65 compiler

users: C/65 owners only

usage: C65 source-file destination-file [-T]

arguments: two file specifiers

option: -T include C/65 source Text in assembler output

Description:

This command loads and executes the file C65.COM, the OSS small-C compilier. Two filenames are required. The first file given must be the name of a text file containing C/65 source code and statements. The second file specified will be created (or reused, if it already exists), and the compiler will write MAC/65-compatible assembly language to it.

Option

If the -T option is specified, the MAC/65 file will contain the user's C/65 text lines. Each source line precedes the assembly code it generates, if any.

*** FOR MORE INFORMATION, SEE YOUR C/65 MANUAL ***

Section 5.4

command: CLRDSK

purpose: To initialize a diskette like the Atari 810 disk drive does.

users: Non-Atari disk drive users with OS/A+ Version 2

usage: CLRDSK

arguments: none

options: none

Description:

This utility is only supplied with OS/A+ Version 2 for non-Atari disk drives. It is used to force your non-Atari disk drive to initialize a diskette just like the Atari 810 disk drive does. Hopefully any program that does not work with a diskette initialized in your non-Atari disk drive will work after you initialize the diskette using the CLRDSK utility.

NOTE: CLRDSK formats the diskette first, then writes zeroes to all sectors execpt the directory, boot and VTOC sectors.

Section 5.5

command: CONFIG

purpose: Allows the user to change the status of a configurable drive

users: OS/A+ users with configurable drives

usage: CONFIG [parm1 parm2 ...] [-N]

arguments: an optional list of paramaters which define the desired status of drives in the system

options: -N no drive configuration table will be displayed

Description

If no parameters are given, this command simply reports the status of all drives currently attached to the Atari computer.

If one or more parameters are given, they are presumed to be requests to configurable disk drives to configure themselves. A parameter consists of a single numeric digit (in the range of 1 to 8) followed by one or two alpha characters (the "Mode"). The digit is presumed to be a disk drive number (corresponding to D1: through D8:). The legal character combinations usable as Modes are as follows:

Mode	Meaning
S	Configure this drive as a Single density, single sided drive.
D	Configure this drive as a Double density, single sided drive.
DD	Configure this drive as a Double density, Double sided drive.

Options

Normally, the CONFIG command will list out the current drive configuration. Using the -N option will cause this table to be omitted.

Example:

```
CONFIG 1D 2DD
        requests that D1: be configured as double
        density, single sided, while D2: will
        become double density, double sided.
```

NOTES:

If a configuration request is made, the file manager system is reinitialized and the system status is reported, as if the command CONFIG with no parameters had been given.

If a configuration request is invalid (e.g., if the drive is not capable of being configured via software), the command will report an error.

WARNING: The CONFIG command should be issued BEFORE the ADOS command. If CONFIG is used while ADOS is is installed, the ADOS command MUST be repeated.

Section 5.6

command: COPY

purpose: This program copies files. Note the cautions listed below.

usage: COPY source-file destination-file [-FQSW]
or
COPY file [-FQSW]

arguments: one or two file specifiers

options:

-F	force overwrite of existing file
-Q	query before each file transfer
-S	single disk copy
-W	wait for user response before copying

Description

The copy program copies one or more files without changing the source file. In the first form, all files matching the source-file specifier would be copied to files indicated by the destination specifier, which may be on the same or a different disk. In the second form, the files indicated by the file name would be copied to files having the same name on the same drive. This enables the copying of files on a single disk system. The source and destination file specifiers should be of one of the following forms:

1) [Dn:]file-name
2) Dn:

In form 1, the drive specifier (Dn:) is optional; the current default drive will be assumed if no drive specifier is given. In the second form, all files from the indicated drive would be copied to or from another disk.

Options

The -F option causes the program to overwrite an existing file if it has the same name as a destination file to be copied. If this option is not specified, files whose destination names already exist will not be copied.

The -Q option causes the program to ask the user whether to copy each file.

The -S option indicates to the program that it must perform the copy on a single drive. Copy will prompt the user to insert source and destination disks at the proper time.

The -W option indicates that the program must wait for the user to insert the proper disks before initiating the copy.

CAUTION:

Do NOT use COPY in conjuction with the ADOS command to copy FROM version 2 diskettes TO version 4 diskettes. Instead, use the COPY24 command utility found on your Version 4 Master Diskette.

Examples:

COPY *.*
will copy all files on the current disk on the current drive to another disk on the same drive. The system will prompt the user when the diskette needs to be swapped. Generally, DUPDSK is a more effective and faster means of performing this function.

COPY *.COM D3: -F
will copy all files having an extension of ".COM" from the current disk drive to drive 3 (which could be the same as the current drive; caution). If the file(s) already exist on drive 3, they will be erased and rewritten.

COPY D2:C*.* D1: -Q
will ask the user if he wants to copy each file starting with the letter "C" from drive 2 to drive 1.

COPY D1:TEST D2:NEWTEST
will copy the file TEST on drive 1 to the file NEWTEST on drive 2.

COPY D1:TEST D1:NEWTEST -S
will perform a single disk copy of TEST to NEWTEST.

Section 5.7
---

command: COPY24

purpose: transfer files from version 2 to version 4 diskettes (and vice versa)

users: Atari owners with OS/A+ version 4 only

usage: COPY24 source-file destination-file [-FQWD

arguments: source and destination file specifiers

options:

-F	force overwrite of existing files
-Q	query before each file copy
-W	wait for response before copying
-D	the version 2 disk (whether source or destination) is a Double Density disk

Description:

The COPY24 command allows users to transfer files between the version 2 (or Atari DOS) and version 4 file systems. In order to use this utility, the disk drives must have already been places into the proper mode using the CONFIG command (single drive users ignore this step). Then the ADOS command must be issued to allow OS/A+ to access version 2 diskettes. At this point, the COPY24 command may be issued.

In order to specify the direction of file transfer, the file specifier of the version 2 files should be of the form An:filename, while the version 4 specifier should be of the form Dn:filename. If the source drive is the same as the destination drive, COPY24 will prompt for the user to swap disks as necessary.

Options:

The "-F", "-Q", and "-W" options function exactly as they do in the COPY command (see the preceding section on the COPY command).

The "-D" option may be used when it is desired to specify that the Version 2 diskette was formatted and written as a double density diskette (whether by OS/A+ version 2 or by Atari DOS 2.0 patched to enable double density).

Section 5.7 (COPY24 contd.)

Examples:

The command

```
COPY24  A2:*.*  D1:
```

would copy all files on the version 2 disk in drive two onto a version 4 diskette in drive one.

The comand

```
COPY24  A1:TEST  D1:NEWTEST
```

would copy the file TEST from a single density version two diskette to the file NEWTEST on a version 4 diskette. COPY24 would automatically switch the disk drive unit from single density to double density as needed during the copy process.

The command

```
COPY24  A1:TEST  D1:NEWTEST  -D
```

would function exactly as the previous example, except that the version 2 diskette in this case is specified to be double density by the -D option.

Section 5.8
---

command: DO

purpose: This command allows the user to perform several operations with one command line

usage: DO command[;command;command...]
or
DO

arguments: optionally, a list of commands separated by semi-colons

options: none

Description

This DO command allows the user to issue several commands on one line. These commands are not restricted to OS/A+ intrinsic and extrinsic commands, however. For example, the following DO command would load the BASIC language, enter a previously 'list'ed program, and run the program:

```
DO BASIC;ENTER "D:PROGRAM";RUN
```

In the second form of the DO command, the DO program will prompt the user for a list of commands, one at a time, saving these away for use. The entry of just a carriage return when prompted for a command will cause the entire list of commands to be executed.

The DO command may also be used to run a BASIC program upon booting the system (similar to the AUTORUN.SYS function of Atari DOS) by placing a DO command within the STARTUP.EXC file (see chapter 6 on batch processing). For example, placing the following within the STARTUP.EXC file will cause the BASIC program "MENU" to be run upon booting the system:

```
DO CAR;RUN "D1:MENU"
```

Section 5.9

command: DUPDBL

purpose: This program provides fast copying of entire double density diskettes

users: ONLY Atari owners using version 2 OS/A+ AND ONLY those using double density disks

usage: DUPDBL

arguments: none

options: none

Description

The DUPDBL program will prompt the user for source and destination drives, and will ask whether to format the destination disk. The entire source disk will then be copied to the destination disk in a manner somewhat faster than the copy utility would provide. The two disks, however, MUST be double density OS/A+ diskettes formatted under version 2 of OS/A+ (or Atari DOS 2.ØS as patched for double density). IF the destination drive is the same as the source drive, the program will prompt the user to swap disks during the duplication process.

Section 5.10
---

command: DUPDSK

purpose: This program provides fast copying of entire floppy disks of the same size and type

users: Atari owners: all versions and diskettes EXCEPT double density disks under version 2 OS/A+

usage: DUPDSK

arguments: none

options: none

Description

The DUPDSK program will prompt the user for source and destination drives, and will ask whether to format the destination disk. The entire source disk will then be copied to the destination disk in a manner somewhat faster than the copy utility would provide. The two disks, however, must be of the same size and type. If the destination drive is the same as the source drive, the program will prompt the user to swap disks during the duplication process.

CAUTION: Do NOT attempt to use DUPDSK to duplicate double density diskettes under version 2 of OS/A+. Unpredictable and disastrous results may occur! DO use DUPDBL (see previous section) for this purpose.

Section 5.11
---

command: HELP

purpose: This program provides a MENU of system commands to help the beginning user.

usage: HELP

arguments: none

options: none

Description

Although we firmly believe that the command system of the OS/A+ Console Processor (CP) is superior to a menu approach, we can readily understand how the wealth of flexibility offered may overwhelm the new user. Therefore, we have provided this HELP command which provides menu access to the most frequently used system commands.

To use the menu, simply type HELP (followed by a RETURN) any time the CP system prompt appears (usually D1:, followed by the cursor).

The available options are numbered from 1 to 9. To choose an option, simply type a digit from 1 to 9, followed by a RETURN. (Any invalid choice will exit the menu, back to OS/A+.)

Note that each of these options (except number 9) are exactly equivalent to an OS/A+ CP command. In the list which follows, the menu option is followed by its OS/A+ equivalent and a short description of its effect.

1. CAR — Runs any cartridge plugged into the left cartridge slot. Always does a "cold start" of the cartridge.

2. COPY — Allows the user to specify two filenames. Will copy a file from the "FROM" file to the "TO" file. [Allows use of ambiguous (wild card) names. Use "*" and "?" in filenames with caution.]

3. DIR — Allows the user to specify a filename (including an ambiguous filename with "*" or "?") and then lists all files which match the given name. If just RETURN is given instead of a filename, will list all files on the "current" disk drive. [See section 2.2 for info on how to change the "current" disk.]

4. COPY — "Duplicate a File". Special access into the COPY utility to copy a single file using a single disk drive. Not as fast as option 2, but for use on systems with only one disk drive.

5. ERA - Allows the user to specify a filename. Erases the named file if it is on the current disk and if it is not protected.

6. PRO — Allows the user to specify a filename. Sets the system status for the named file to "Protected" if the file exists. [Protected files cannot be ERAsed or written to.]

7. REN — Allows the user to specify a FROM name and a TO filename. RENames the FROM file (if it exists and is not protected) to the TO filename (if it does not already exist).

8. UNP — Allows the user to specify a filename. Resets the system "protected" status for the named file if it exists.

9. — Exit to OS/A+. An unnecessary option, actually, since a simple RETURN will exit also.

Section 5.12

command: INIT

purpose: This program initializes floppy disks so that they may be read from or written to

usage: INIT

arguments: none

options: none

Description

The INIT utility allows the user to format a floppy disk so that it may be read or written by programs. Under OS/A+ version 2, the user will be prompted for information on exactly how to initialize the disk (i.e., with or without a system file, etc.). Under version 4, a system file, DOS.SYS, is not written by the init program. In either case, the program will prompt for a drive to init. When the initialization process is complete, the floppy disk may now be used to store data. Under OS/A+ version 4, use COPY or DUPDSK to transfer a DOS.SYS file to the new disk, if desired.

Section 5.13

command: INITDBL

purpose: This utility is used to initialize a double density diskette so that they can be read from and written to in double density.

users: OS/A+ version 2 users with a single non-Atari disk drive.

usage INITDBL

arguments: none

options: none

Description:

NOTE: The INITDBL utility is unnecessary for users with more than one disk drive. Instead, just use the standard INIT utility (see section 5.12).

The utility INITDBL is to be used on a one drive system to initialize a double density diskette and write DOS.SYS to it.

To use this utility, boot the master diskette. Type the INITDBL command, and answer the prompt with the number 1. Before you type [RETURN], replace the master diskette with your new unformatted double density diskette. When the INITDBL utility is finished the disk drive with still be configured single density. To get a directory of your new double density diskette, CONFIGure the disk drive to double density and type the DIR command.

Section 5.14

command: MAC65

purpose: Loads and executes the MAC/65 macro assembler

users: MAC/65 disk owners only

usage: MAC65 [file1 [file2 [file3]] [-A][-D]]

arguments: an optional set of one to three filename, construed to be the source, listing, and object files (respectively) of a MAC/65 assembly.

options:

-A	source file is Ascii
-D	assembly must be Disk-to-Disk

Description:

This command loads and executes the file MAC65.COM, the OSS Macro Assembler/Editor. If no filenames are given, MAC/65 will be invoked in its interactive (Editor) mode. Programs or text may then be edited and/or assembled. See the MAC/65 manual for further details.

If one or more files are specified, MAC/65 will be invoked in its "batch" mode. That is, it will perform a single assembly and then return to OS/A+. Generally, this command line will perform the assembly in a manner equivalent to giving the "ASM" command from the MAC/65 Editor. That is, if only one filename is given, it is assumed to be the source file, implying that the listing will go to the screen and the object code will be placed in memory (but only if requested by the .OPT OBJ directive). If a second filename is given, it is assumed to be the name of the listing file. Only if all three filenames are given will the object code be directed to the file specified.

NOTE: if an assembly needs no listing but does need an object file, the user may specify E: as the listing file, thus sending the listing to the screen.

Options

The -A option is used to specify that the source file is not a standard MAC/65 SAVEd file but is instead an Ascii (or Atascii) file. This is equivalent to using the interactive Editor mode of MAC/65 to use the sequence of commands "ENTER#D..." and "ASM ,...".

The -D option is used to specify that the assembly MUST proceed from disk to disk. If this option is not given, the source file is LOADed (or ENTERed) before the assembly, and then the assembly proceeds with the source in memory (generally producing improved speed of assembly). If, however, the source file is too large to be assembled in memory, the user may use this option to allow assembly of even very large programs. (And remember, even if the source fits, the macro and symbol tables must reside in memory during assembly also.)

NOTE: the -D option can NOT be used in conjunction with the -A option. The source file assembled under the -D option MUST be a properly SAVEd (tokenized) file.

*** FOR MORE INFORMATION, SEE YOUR MAC/65 MANUAL ***

Section 5.15
---

command: RS232

purpose: installs the serial device handlers ("Rn:") for use with the Atari 850 Interface Module.

users: Atari users with 850 Modules

usage: RS232

arguments: none

options: none

Description:

Using the command RS232 from OS/A+ is functionally equivalent to using Atari's AUTORUN.SYS file (which boots the R: handlers at power on time under Atari DOS). The driver for the various RS232 functions is loaded at LOMEM, LOMEM is moved, and the R: device is hooked into the handler table.

After giving the RS232 command, if the Dn: prompt reappears BELOW the line containing the "RS232" command, the Interface Module has loaded its software properly. If, however, the screen clears and the Dn: prompt appears at the TOP of the screen, something went wrong during the loading process. Unfortunately, the software in the Interface Module does not return a usable error code, preferring instead to do a system warmstart (hence the cleared screen).

CAUTION: due to a bug in the software in the 850 Interface Module, hitting RESET will destroy the proper LOMEM pointer, effectively ignoring the space occupied by the RS232 handlers.

CAUTION: the 850 Interface Module is sometimes too intelligent for its own good. In particular, one cannot generally reload the software from the module without turning the module off and back on again.

Section 5.16
---

command: SDCOPY

purpose: This program is used to copy single density files to double density files.

users: Atari owners using OS/A+ version 2 only.

usage: SDCOPY source-file destination-file [-FQR

arguments: one or two file specifers

options:

-F	force overwrite of existing file
-Q	query before each file transfer
-R	reverse orientation of copy
-V	verbose

Description:

The utility SDCOPY is for OS/A+ Version 2 owners with non-Atari disk drives only. The purpose of this utility is to copy a single density source file to a double density destination file. This utility works the same way as the COPY utility execpt for the -R option.

The -R option is used to reverse the orientation of the copy. That is instead of copying from a single density source file to a double density destination, the orientation is reversed and the copy goes from a double density source file to a single density destination file.

NOTE: the SDCOPY utility can only be used with one disk drive. If you want to copy from single density to double density between two different drives, just use the CONFIG command to set the drives up properly and use the normal COPY utility.

Chapter 6 Batch Processing

6.1 An Overview of Batch Processing

You may often find yourself repeating the same group of commands over and over. OS/A+ allows you to put these commands into a file with special capabilities.This file may be used by typing a single command which will cause all the commands in that file to be executed. This can save quite a bit of your time and energy since you won't constantly be typing the same string of commands.

Let's suppose that you wrote a set of BASIC programs that had to be run in sequence. You could do this in two ways:

1. Issue the CP extrinsic BASIC command (thus executing BASIC.COM), then from BASIC run each program one at a time. If the running time of the BASIC programs was very long you might sit at the key board for hours just to type RUN every once in awhile.

OR

2. Create a BATCH file containing the OS/A+ commands required to run the BASIC programs. You would then enter one command that would free you from the keyboard for more important (or fun) things.

The second method is obviously preferable as it is quicker and can be repeated easily.

Any text file with the filename extension .EXC can be used as an OS/A+ batch execute file. The execution of the file is invoked much like the extrinsic commands, except the command is preceeded with a commercial "at" symbol ("@"). To execute the EXECUTE file DEMO.EXC on the D1: default device, type:

```
D1:@DEMO
   -----
```

CP will open the file spec D1:DEMO.EXC for input and then set up OS/A+ to read it line by line executing the CP commands just as if they were being entered from the keyboard.

6.2 .EXC File Format

An execute file is simply a text file. Each line of this text file will become a CP command when executed.

The three basic rules of the text file lines are:

1) they must contain valid OS/A+ Console Processor commands
2) they must be shorter than 128 characters in length
3) they must end in a carriage return (ATASCII $9B).

OS/A+ allows the commands in an execute file to be preceeded by numbers and blanks. This feature allows the command lines to be numbered for readability and to document their purposes.

The command file lines: LOAD OBJ.TEST <return>
and
100 LOAD OBJ.TEST <return>

are the same to OS/A+ . OS/A+ scans the line for the first non- numeric, non-blank character before starting to scan the command word. Virtually any text editor, including the editor of MAC/65, can be used to create and modify execute files.

NOTE: One may also create an execute file (or, for that matter, any text file) by using "TYP E: <diskfile>". (TYPE will clear the screen, at which time you simply type in your text, line by line. You terminate the copy by pressing CTRL-3 on the Atari, the end of file signal for the E: device.)

6.3 Intrinsic Commands for .EXC Files

OS/A+ has four special intrinsic commands designed for use exclusively with execute files. These commands are:

REMARK	Remark or comment (does nothing)
SCREEN	Turn on Echo of execute file command lines to the screen. (Default mode)
NOSCREEN	Turn off Echo of execute file command lines
END	Stop executing the execute file and return OS/A+ to keyboard entry mode (the CP).

See the section on intrinsic commands for a more detailed explanation of these commands.

6.4 Stopping Batch Files

While an execute file is being processed, various conditions may occur which will warrant a halt in the batch execution. These conditions may occur because of system-detected errors or because of a user program detecting a condition it considers hazardous to the system's health.

6.4.1 Stops by OS/A+

Humans are not quite perfect in the eyes of computers and sometimes make mistakes. OS/A+ commands specified in error will generate error messages. If OS/A+ discovers an error while executing an EXECUTE file, it will print the error message as usual and STOP executing the EXECUTE file. Note that this error stop only occurs if the error is found by OS/A+, not just because a program generates an error.

Execution of an execute file will also stop after the CARTRIDGE commmand is executed.

6.4.2 Stops by User Programs

It is sometimes desirable for a program in a chain of executing programs to stop the execute process. The usual reason for this is that the program has detected an error severe enough to invalidate the processes performed by the following program(s). The continued execution of the execute files is provided for by a single byte flag within OS/A+. If a program sets this byte to zero, then upon returning to OS/A+ (DOS or CP BASIC statements) the execute file execution will immediately stop. The execute flag is located 12 bytes from the start of OS/A+, which is pointed to by memory location 10 ($0A). The following Basic program segment will turn off the execute file and return to OS/A+.

```
1000      CPADR = PEEK(11)*256 + PEEK (10)
1010      EXCFLG = CPADR + 12
1020      POKE  EXCFLG,0
1030      DOS
```

6.5 STARTUP.EXC: A Special File

The execute filename STARTUP.EXC has special meanings in the OS/A+ system. When the system is first booted (power up), OS/A+ will search the directory of the booted disk volume for a file named STARTUP.EXC. If STARTUP.EXC is on the booted volume, OS/A+ will execute that file before requesting keyboard commands.

CHAPTER 7

GETTING STARTED WITH OS/A+ and Atari BASIC

Now we are really ready to start explaining some of the OS/A+ capabilities. But first, let's get the hardware set up right and things put in their proper places. Right off, put the master diskette in a safe place, take the copied master diskette and put that in drive 1, put your BASIC cartridge in the left cartridge slot, and turn the power ON again.

Since you sucessfully made a copy of the master disk, we'll assume that the system booted properly (if it didn't, try everything again with a new blank diskette before calling us).

Now, though, when the system finally presents you with the HELP menu, you can use selection 1, "enter CARtridge". If you do so, you should see BASIC's familiar READY prompt, and all is well. (Incidentally, you could also have used selection 9 to go to the CP command level and then issued the CAR command. More on this soon.)

Suppose, though, that you wanted to return to the CP command level. Why do that? Well from the CP you can list all files on a diskette via DIRectory, RENname files, PROtect files and so on, without losing any BASIC program that you might have already typed in.

In fact any of the intrinsic commands (found in chapter 4 of the OS/A+ manual) may be used from the CP without harming even a line of what you may have typed in while in BASIC. To get back to your program just type CAR. You will be back in BASIC, and any program you in is still there.

So, from the D1: prompt in OS/A+, CAR will put you in BASIC (make sure the cartridge is there first). Nice and neat. But how do we get from BASIC to the CP? Simple: From the READY prompt in BASIC, the DOS command will put you back in the OS, where you can do any intrinsic commands without losing any of the BASIC statements you might have already typed in.

The following sections describe the most common BASIC commands and statements which affect files on the disk. Please note that these commands should be issued while using the BASIC cartridge, or OSS BASIC. That is , these commands should be typed immediately after the READY prompt.

Section 7.1
---

command: CLOSE

purpose: This command disassociates the file number (channel) and file which were associated by a previous OPEN statement.

users: Atari BASIC users with OS/A+

usage CLOSE #fn

argument: fn - file number 1-7

description:

After CLOSEing a file number, the user may no longer perform I/O (e.g, via PRINT, INPUT, etc.) on the file which had been associated with that channel.

NOTE: a file OPENed for any form of output (modes 8,9, or 12) should ALWAYS be closed before the diskette containing it is removed or changed. The most common cause of crashed Atari Diskettes is failure to observe this rule.

NOTE: Atari BASIC does NOT consider it an error to CLOSE a channel that is not OPEN, so it is often good practice to end a program segment by a line such as the following:

```
999 FOR I=1 TO 7 : CLOSE #I : NEXT I
```

NOTE: both the END and RUN statements close all files (except file #Ø, the keyboard/screen), and can be used to advantage for this purpose when desired.

Section 7.2
---

command: ENTER

purpose: This command is used to retrieve a BASIC program that has been LISTed to the disk.

users: Atari BASIC users with OS/A+

usage: ENTER filespec

argument: filespec - the name of the file you are going to ENTER.

description:

The ENTER command is used to retrieve a BASIC program that has been LISTed to the disk. As the program is being ENTERed into BASIC's user area, each line will be checked for proper syntax and converted into the internal (tokenized) form used by BASIC.

If a syntax error is encountered, the offending line will be listed with the suspected error location in inverse video.

NOTE: The line with the error will, nevertheless, be placed in program memory. In such a case, your program must be corrected before you can RUN it.

CAUTION: ENTER does NOT clear the user memory space. Therefore, if you wish to ENTER a new program, use NEW first. (Actually, this can be a handy feature when you wish to merge two programs together.)

EXAMPLE

```
10 PRINT  "THIS IS PROGRAM 1"
LIST "D:PROG1"
10 PRINT  "THIS IS PROGRAM 2"
LIST "D:PROG2"
NEW
ENTER "D:PROG1"
LIST
[and the computer will LIST the following:
        10 PRINT  "THIS IS PROGRAM 1" ]
NEW
ENTER "D:PROG2"
RUN
[and the computer will respond with:
        THIS IS PROGRAM 2   ]
```

Section 7.3
---

command: GET

purpose: This statement will retrieve a single byte of data from a specified disk file.

users: Atari BASIC users with OS/A+

usage: GET #fn,avar

arguments: fn - file number 1-7
avar - any numeric variable

description:

The GET statement is used to retrieve a single byte of data from a disk file that has been previously OPENed using the same file number.

NOTE: The data that you are GETting from the disk file should have been previously written to the specified file using the PUT statement.

EXAMPLE:

```
10 OPEN #1,8,0,"D:TEST" : REM CREATE A TEST FILE
20 FOR I = 0 TO 255 : PUT #1,I :NEXT I
30 CLOSE #1 :REM WE CREATED IT
40 OPEN #1,4,0,"D:TEST" : REM NOW CHECK IT OUT
50 FOR I = 0 TO 255 : GET #1,X : REM CHECK EACH
60   IF X <> I THEN PRINT "BAD DISK DATA",I,X
70 NEXT I
80 END : REM END CLOSES ALL FILES
```

Section 7.4
---

command: INPUT

purpose: This command is used to request data from the specified file number (or keyboard).

users: Atari BASIC users with OS/A+

usage: INPUT {#fn,} var {,var...}

arguments: fn - file number 1-7
var - either numeric or string

description:

When the INPUT statement is used without the fn option, data will be requested from the keyboard. You will notice a "?" appearing on the screen prompting you for the keyboard input. See your Atari BASIC Reference Manual for more details.

When the file number (#fn) argument is used, data will come in the form of ATASCII lines from the file that has been previously successfully OPENed using the same file number. Otherwise, the action of INPUT is virtually identical to the action when INPUTing data from the keyboard. That is, a string input is terminated by an ATASCII RETURN character and a numeric input by either the RETURN or a comma within a line.

NOTE: The INPUT statement cannot (generally) read a line that is longer the 127 characters in length. If you PRINT a line to the disk that you will later want to INPUT, it is best to limit the size of the PRINTed line to 127 characters or less.

EXAMPLE

```
10 DIM LINE$(15)
20 OPEN #1,8,0,"D1:INPUT.SMP" : REM CREATE A FIL
30 FOR I = 1 TO 20
40 PRINT #1;"THIS IS LINE #";I : REM WRITE THE D
50 NEXT I
60 CLOSE #1 : REM CLOSE THE FILE YOU JUST CREATE
70 OPEN #1,4,0,"D1:INPUT.SMP" : OPEN FOR READ ON
80 FOR I = 1 TO 20
90 INPUT #1,LINE$ : REM GET THE FIRST LINE
100 IF LINE$(15) <> STR$(I) THEN GOTO 500
110 PRINT LINE$
120 NEXT I
130 CLOSE #1 : REM CLOSE THE FILE
140 PRINT "SUCESSFUL USE OF THE INPUT STMT"
150 STOP
500 REM WE GET HERE FROM LINE 100
510 PRINT "UNSUCCESSFUL USE OF INPUT"
520 END : REM ANOTHER WAY TO CLOSE THE FILE
```

Section 7.5
---

command: LIST

purpose: This command will LIST the program currently in memory to the screen (or to the file specified).

users: Atari BASIC users

usage: LIST [filespec]
LIST [filespec,] linenol [,lineno2]

arguments: filespec - the name of the file you are going to LIST to the disk.
linenol - beginning line number
lineno2 - ending line number

description

The LIST command is probably one of the most commonly used commands in BASIC. Most people know that the LIST command, when given all by itself, will LIST their program to the screen. Even when beginning and ending line numbers are given the results are predictable.

Now with OS/A+ the LIST command can do even more. When used with a filespec, the LIST command will LIST your program to the disk instead of the screen. The contents of this file will contain text characters and can take up a large amount of disk space if you have a large program.

If you use the option where two line numbers are given, then only the lines from linenol to lineno2 (inclusive) will be LISTed to the filespec.

If you use the option where only one line number is given, then ONLY that line will be LISTed to the filespec.

NOTE: The ability to LIST a range of lines to the disk provides a convenient method of moving a subroutine (for example) to another program.

See also Section 7.2 on the ENTER command.

Section 7.6
---

command: LOAD

purpose: This command will get a program that has been SAVEd to the disk and put it in BASIC's memory.

users: Atari BASIC users with OS/A+ or DOS 2.0s

usage: LOAD filespec

arguments: filespec - The name of the file you wish to LOAD.

description:

LOAD is used in conjunction with the BASIC SAVE command. Only programs which have been previously SAVEd to disk may be LOADed. No syntax checking will be done as your program is being LOADed, because the program is already in internal format.

Generally, if you wish to keep a program on the disk, you SAVE it. Then, later, when you wish to look at it, modify it, or RUN it, you can LOAD it. BASIC does not remember the name that you use when you LOAD a program, so you can SAVE it again either under the same name (in which case the original version is lost) or under another name.

Also, see the RUN command for an alternative method of LOADing a program which will simply be RUN and not modified.

EXAMPLE:

```
10 PRINT "THIS IS PROGRAM 1"
SAVE "D:PROG1"
10 PRINT "THIS IS PROGRAM 2"
SAVE "D:PROG2"
LOAD "D:PROG1"
LIST
[and the computer will list the following:
      10 PRINT "THIS IS PROGRAM 1" ]
RUN "D:PROG2"
[and the computer will respond with:
      THIS IS PROGRAM 2   ]
```

Section 7.7
---

command: OPEN

purpose: This command prepares a file for access and assigns it a file number.

users: Atari Basic users with OS/A+

usage: OPEN #fn,aexp1,aexp2,filespec

arguments:

```
fn     - file number 1-7
aexp1  - I/O mode
           4 - input
           6 - directory access
           8 - output
           9 - append
          12 - input/output
aexp2      - device dependent value
             (usually Ø)
filespec   - a proper OS/A+ filename
```

description:

The OPEN statement allows a disk file (or any device, for that matter) to be linked to a file number (channel) for future reference in connection with file input/output instructions (e.g., PUT,GET,INPUT,PRINT, CLOSE).

COMMENTS on arguments:

The fn argument allows for a number between 1 and 7. The number Ø is reserved for the screen and can not be used in Atari BASIC (though it is allowed in BASIC A+). After a file has been OPENed with a given fn, all references to that file must be made using that same fn.

The aexp1 argument allows the user to OPEN a file for a specific "mode", according to the following table:

Mode 4: will OPEN the specified file for input only. Thus you can only retrieve data from the specified file.

Mode 6: allows you to access the directory on the disk.

Mode 8: is the opposite of mode 4. That is, data can only be stored to the specified file. See below for notes when using mode 8.

Mode 9: is used to add data to the specified file. The data that is added will begin at the current end of the specified file.

Mode 12: is used to access the specified file for input AND output. Thus data can be stored and retrieved from the specified file.

NOTE: After OPENing a file, the specified file number is used to designate the file in other I/O statements. Two OPENed files cannot have the same file number, but it is possible to OPEN the same file with two different file numbers. Generally, such a double OPEN will have disastrous results. BEWARE:

NOTE: If a file is OPENed for output (aexpl=8) and the specified file does not exist then a file with the specified name will be created for you. If the file specified already exists, it will be destroyed and a new file with the specified name will be created for you.

NOTE: A file OPENed for update (aexpl=12) can NOT be appended to under version 2 of OS/A+ or under Atari DOS. Excepting for version 4 OS/A+ files, only files opened in mode 9 will allow a file's size to be increased.

NOTE: Modes 13 and 5 are also possible under version 4 OS/A+. Their usage, however, is best left to advanced users; refer to section 8.2.2 for more details of extended meanings of aexpl and aexp2 under version 4.

NOTE: Mode 6 might, for example, be used from BASIC to find what files are on a disk and thereby allow a menu selection. The following program will allow a menu selection of all BASIC SAVEd programs on drive 1, providing that the program names do NOT have an extension (i.e., the programs should not have been SAVEd as "D:name.ext" but simply as "D:name").

EXAMPLE:

```
100 OPEN #1,6,0,"D:*" : DIM LN$(40)
110 FOR I = 1 TO 20 : INPUT #1, LN$
120 IF LN$(2,2)=" " THEN PRINT I,LN$(3,10) : NEXT I
130 CLOSE #1 : OPEN #1,6,0,"D:*"
140 PRINT : PRINT "WHAT PROGRAM TO RUN ";
150 INPUT J : IF J>=I THEN GOTO 140
160 FOR I = 1 TO J : INPUT #1,LN$ : NEXT I
170 CLOSE #1 : LN$(1,2) = "D:"
180 RUN LN$(1,10)
```

Try typing this in and then saying SAVE "D:MENU".
Later, you can use the program by typing RUN "D:MENU".

Section 7.8
---

command: PRINT

purpose: This command puts the ASCII equivalents of the given expressions to the file specified or the screen.

users: Atari BASIC users with OS/A+

usage: PRINT [#fn {;}] exp [{,}exp...] {,}

arguments:
fn - file number 1-7
exp - the expression can either be a string enclosed in double quotes, a string variable, or a numeric variable.

description:

When a file number is used with the PRINT command, the specified variables are PRINTed to the disk file that has been previouly OPENed using the same file number.

NOTE: Characters are PRINTed to a disk file in a manner identical to the way characters are PRINTed to the screen if the file number option is not used.

NOTE: A "," after the #fn causes tabbing before the first character is PRINTed. A ";" does not cause the tabbing. Normanlly, the semicolon should be used.

See also INPUT.

Section 7.9
---

command: PUT

purpose:
this statement is used to store a single byte of data to a specified file

users: Atari BASIC users with OS/A+

usage: PUT #fn,avar

arguments: fn - file number 1-7
avar - an arithmetic varaible

description:

The PUT statement is used to output a single byte of data to a specified file. The file number used in the PUT statement must be one that has been previously used in the successful OPEN of a file.

NOTE: Data that has been stored in a file using the PUT statement can usually only be retrieved using the GET statement.

See also GET.

Section 7.10
---

command: SAVE

purpose:
This command will store a BASIC program on disk in internal format (not ATASCII).

users Atari BASIC with OS/A+

usage: SAVE filespec

arguments: filespec - filename you wish to SAVE you program under.

description

The SAVE command is used to SAVE your BASIC program in its internal format. This format is usually smaller then the text form of your program and will take up less room on your disk. All programs SAVEd to the disk must be reentered using the LOAD or RUN commands.

See descriptions of LOAD and RUN for more examples and further explanations.

Section 7.11
---

command: XIO

purpose: This is BASIC's catch-all Input/Output command. If BASIC doesn't provide a function to access a particular feature of a device or file, some form of XIO can probably be used to do so.

users: Atari BASIC users with OS/A+.

usage: XIO subcommand, #fn, aux1, aux2, filespec

arguments: subcommand -- see descriptions below.

fn -- a file number. In contrast to most OS/A+ I/O commands, XIO often requires that the file number be that of an UN-OPENed channel. The subcommand dictates the usage here, so see descriptions below.

aux1 and aux2 -- generally zero. These values are passed to OS/A+ unchanged (and thence to the device being accessed), so the individual device(s) may require other values. None of the examples given in this section use these values.

filespec -- a proper OS/A+ file name.

descriptions:

Although, as noted, XIO can be used for several purposes, we will restrict our discussion here to those four subcommands most useful to the Atari BASIC programmer. For more detail, we suggest chapter 5 of the OS/A+ reference manual and other sources, such as the Atari 850 Interface Module manual.

The subcommands to be discussed will each be treated as a separate BASIC command.

subcommand: RENAME

purpose: May be used to rename disk files.

usage: XIO 32, #fn, Ø, Ø, filespec

arguments: fn -- the file number of an UN-OPENed channel.

filespec -- a proper OS/A+ file name followed by, in the same BASIC string, a comma and a second file name. The second file name may NOT include a disk drive specifier.

description:

It is suggested that "fn", the file number, be 7, since that channel is normally reserved for system I/O functions (which this certainly is). The only thing strange about this subcommand is the form of the filespec. Some examples follow:

```
XIO 32,#7,Ø,Ø,"D:TEST.SAV,OLDTEST.SAV"
DIM FL$(1ØØ)
INPUT FL$
FL$(LEN(FL$)+1) = ",BACKUP"
XIO 32,#7,Ø,Ø,FL$
```

Again, note that the second file name in both examples is NOT preceded by a disk drive specifier.

subcommand: ERASE (also called "KILL" and "DELETE")

purpose: May be used to permanently erase disk files.

usage: XIO 33, #fn, Ø, Ø, filespec

arguments: fn -- the file number of an UN-OPENed channel.

filespec -- a proper OS/A+ file name, with "wild cards" accepted and processed.

description:

IF the file specified exists on the disk drive specified, and IF the file is not PROTECTED (see next subcommand), the specified file will be permanently erased (deleted, killed, zapped) from the disk.

USE THIS SUBCOMMAND WITH CAUTION: specifying a "wild card" (a file name including an asterisk or question mark) will erase ALL files which match the given name.

Examples:

XIO 33,#7,Ø,Ø,"D2:OLDPROG.SAV"
will erase the single file with the name OLDPROG.SAV from the diskette in drive 2.

XIO 33,#5,Ø,Ø,"D:*.BAK"
will erase all files having a filename extension of ".BAK" from the diskette in drive 1.

subcommand: PROTECT (also called "LOCK")

purpose: May be used to protect disk files from accidental erasure and modification.

usage: XIO 35, #fn, Ø, Ø, filespec

arguments: fn -- the file number of an UN-OPENed channel.

filespec -- a proper OS/A+ file name, with "wild cards" accepted and processed.

description:

All files on the specified drive which have names which match the specified file will be "PROTECTED" by usage of this subcommand. Protection in the OS/A+ environment simply consists of setting a flag in the diskette's file directory which tells the OS to disallow either modification (i.e., OPENs in modes 8, 9, 12, etc.) or erasure of the file. Any OS/A+ DIRectory listing will show protected files by means of an asterisk in the first column of the displayed lines (unprotected files have simply a space in that position).

Examples:

```
XIO 35, #7, Ø, Ø, "D:*.*"
        will protect ALL files on drive 1.
XIO 35, #1, Ø, Ø, "D4:DOS.SYS"
        will protect only the file named
        "DOS.SYS" on the diskette in drive 4.
```

subcommand: UNPROTECT (also called "UNLOCK")

purpose: May be used to unprotect disk files to allow subsequent erasure and modification.

usage: XIO 36, #fn, Ø, Ø, filespec

arguments: fn -- the file number of an UN-OPENed channel.

filespec -- a proper OS/A+ file name, with "wild cards" accepted and processed.

description:

All files on the specified drive which have names which match the specified file will be "UNPROTECTED" by usage of this subcommand. Protection in the OS/A+ environment simply consists of setting a flag in the diskette's file directory which tells the OS to disallow either modification (i.e., OPENs in modes 8, 9, 12, etc.) or erasure of the file. Any OS/A+ DIRectory listing will show UNprotected files by means of a space in the first column of the displayed lines (protected files have an asterisk in that position).

Examples:

XIO 35, #7, Ø, Ø, "D2:*.COM"
will unprotect all files on drive 1 which have a filename extension of ".COM".

XIO 35, #1, Ø, Ø, "D1:DOS.SYS"
will unprotect only the file named "DOS.SYS" on the diskette in drive 1 (this step is necessary before erasing that file, as you might do to gain more space on the diskette).

Chapter 8: Assembly Language and OS/A+

As mentioned in Section 2.1, OS/A+ is designed as a layered operating system. Application programs (including languages such as BASIC A+) are expected to call the operating system "properly", through the system call vector (labeled "CIO" in SYSEQU.ASM). In turn, the CIO will determine which device is to receive what I/O request and handles most of the work transparent to the calling program.

If a program restricts itself to proper calls to CIO using labels provided in SYSEQU.ASM, the program should transfer virtually without change from one version of OS/A+ to another. (Probably the only other areas of change would involve memory map usage and the length of file names--12 bytes under version 2 and 30 bytes under version 4.)

In any case, here with is a description of the proper assembly language calling sequnces and parameters under OS/A+.

8.1 Interfacing to I/O Routines

8.1.1 The Structure of the IOCB's

When a program calls the OS through location "CIO", OS expects to be given the address of a properly formatted IOCB (Input Output Control Block). For simplicity, we have predefined 8 IOCB's, each 16 bytes long, and the calling program specifies which one to use by passing the IOCB number times 16 in the 65Ø2's X-register. Thus, to access IOCB number four, the X-register should contain $4Ø on entry to OS. Notice that the IOCB number corresponds directly to the file number in BASIC (as in PRINT #6, etc.). The IOCB's are located from $Ø34Ø to $Ø3BF on the Atari (but you really should use the equates from the disk file "SYSEQU.ASM" rather than relying on hard-coded addresses.)

When the OS gets control, it uses the X-register to inspect the appropriate IOCB and determine just what it was that the user wanted done. Figure 8-1 gives the OS/A+ standard name for each field in the IOCB along with a short description of the purpose of the field. Study the figure before proceeding.

The user program should NEVER touch fields ICHID,ICDNO, ICSTA and ICPUT, as they are set by the OS. In addition, unless the particular device and I/O request requires it, the program should not change ICAUX1 through ICAUX6. The most important field is the one-byte command code, ICCOM, which tells the operating system what function is desired.

FIGURE 8-1

IOCB STRUCTURE

FIELD NAME	OFFSET WITHIN IOCB (bytes)	SIZE OF FIELD (bytes)	PURPOSE OF FIELD
ICHID	Ø	1	SET BY OS. Index into device name table for currently OPEN file, set to $FF if no file open on this IOCB.
ICDNO	1	1	SET BY OS. Device number (e.g., 1 for "D1:xxx" or 2 for "D2:yyy")
ICCOM	2	1	The COMMAND request from user program. Defines how rest of IOCB is formatted.
ICSTA	3	1	SET BY OS. Last status returned by device. Not necessarily the status returned via STATUS command request.
ICBADR	4	2	BUFFER ADDRESS. A two byte address in normal 65Ø2 low/high order. Specifies address of buffer for data transfer or address of filename for OPEN, STATUS, etc.
ICPUT	6	2	SET BY OS. Address minus one of device's put-one-byte routine. Possibly useful when high speed single byte transfers are needed.
ICBLEN	8	2	BUFFER LENGTH. Specifies maximum number of bytes to transfer for PUT/GET operations. NOTE: this length is decremented by one for each byte transfered.

ICAUX1	10	1	Auxiliary byte number one. Used in OPEN to specify kind of file access needed. Some drivers can make additional use of this byte.
ICAUX2	11	1	Auxilliary byte number two. Some serial port functions may use this byte. This and all following AUX bytes are for special use by each device driver.
ICAUX3 ICAUX4	12	2	For disk files only: where the disk sector number is passed by NOTE and POINT. (These bytes could be used separately by other drivers.
ICAUX5	14	1	For disk files only: the byte-within-sector number passed by NOTE and POINT.
ICAUX6	15	1	A spare auxilliary byte.

FIGURE 8-1

IOCB STRUCTURE

IOCB field name	Ø	1	2	3	4	5	6	7
					BUFFER		PUT-A-	
	I	I	I	I	ADDRESS		.BYTE	
	C	C	C	C			ADDRESS	
Type of	H	D	C	S				
command	I	N	O	T	ICBADR		ICPUT	
OPeN	*	*	3	*	filename		*	
CLOSE	*		12	*				
dynamic STATus		*	13	*	filename			
Get TeXT Record			5	*	buffer			
Put TeXT Record			9	*	buffer			
Get BINary Record			7	*	buffer			
Put BINary Record			11	*	buffer			
EXTENDED COMMANDS: DISK FILE MANGER ONLY								
RENname		*	32	*	filename			
ERAse		*	33	*	filename			
PROtect		*	35	*	filename			
UNProtect		*	36	*	filename			
NOTE			38	*				
POINT			37	*				

LEGEND:

'*'	Set by OS when this command is used
'buffer'	Address of a data buffer
'filename'	Address of a filename

Figure 8-2 IOCB Field Usage

8 / 9	10	11	12	13	14	15	IOCB field name
BUFFER LENGTH / ICBLEN	ICAUX1	ICAUX2	ICAUX3	ICAUX4	ICAUX5	ICAUX6	(as given in SYSEQU.ASM) COMMAND NAMES
	mode						COPN
							CCLOSE
							CSTAT
length							CGTXTR
length							CPTXTR
length							CGBINR
length							CPBINR
(See section 8.1.2)							
							CREN
							CERA
							CPRO
							CUNP
			sec num	byte			CNOTE
			sec num	byte			CPOINT

'length' Length of a data buffer
'mode' Mode of OPEN (i.e., read, write, etc.)
'sec num' Sector number, see section 8.1.2
'byte' Byte in sector, see section 8.1.2

Figure 8-2(con't.)

8.1.2 The I/O Commands

Figure 8-2 provides a table of I/O commands and their usage of the various fields of the IOCB's. The first seven are OS/A+ oriented and will be dealt with in part A) of this section. The last six are File Manager specific and are discussed in part B).

Most of the commands manipulate a device in some way, so maybe we should talk about them for a moment. Device names under OS/A+ are very simplistic; they consist of a single letter optionally followed by a single digit used to define a specific device when more than one of the same kind exist (Ex.- D1: or D2:). Traditionally (and, in the case of Atari disk files, of necessity) the device name is followed by a colon. The following devices are implemented under standard OS/A+:

E: The keyboard/screen editor device. The normal console output.

K: The keyboard alone. Use this device to bypass editing of user input.

S: The screen alone. Can be either characters (ala E:) or graphics.

P: On the Atari, the printer. The standard device driver allows only one printer.

C: The cassette recorder.

D: The disk file manager, which also usually requires a file name.

Other device names are possible (e.g., for RS-232 interfaces), and in fact the ease with which other devices may be added is another mark for the claim that OS/A+ is a TRUE operating system. The structure of device drivers is material for a later section (8.3), but we should like to point out that, on the Atari, the OS ROM includes drivers for all the above except the disk. In fact, the drivers account for over 5K bytes of the ROM code. The screen handler, with all its associated editing and GRAPHICS modes, occupies about 3K bytes of that.

A) The Standard OS/A+ Commands

The OS itself only understands a few fundamental commands, but OS/A+ also provides for extended commands necessary to some devices (XIO in BASIC). In any case, each of these fundamental commands deserves a short description.

OPEN

Open a device (synonyms: file, IOCB, channel) for read and/or write access. OS expects ICAUX1 to contain a byte that specifies the mode of access:

ICAUX1	MODE
4	Read Only
5	Read Only Append
6	Read Directory Only
8	Write Only
9	Write Only Append
12	Read/Write(Update)
13	Read/Write Append/Update

NOTE: modes 5 and 13 are only available on version 4 of OS/A+ (more information about them, 6, and 9 may be found in part B). The name of the device (and, for the disk, the file) must be given to OS; this is accomplished by placing the ADDRESS of a string containing the name in ICBADR.

CLOSE

Terminate access to a device/file. Only the command must be given.

STATUS

Request the status of a device/file. The device can interpret this request as it wishes, and pass back a (hopefully) meaningful status. As with OPEN, the ADDRESS of a filename must be placed in ICBADR.

GET TEXT

A powerful command, this causes the OS to retrieve ("GET") bytes one at a time from a device/file already OPENed until either the buffer space provided by the user is exhausted or a RETURN character (Atari $9B) is encountered. The user specifies the buffer to use by placing its ADDRESS in ICBADR and its maximum size (length) in ICBLEN.

PUT TEXT

The analogue of GET TEXT, OS outputs characters one at a time until a RETURN is encountered or the buffer is empty. Requires ICBADR and ICBLEN to be specified.

GET DATA

Extremely flexible command, this causes OS to retrieve, from the device/file previously OPENed, the number of bytes specified by ICBLEN into the buffer specified by ICBADR . NO CHECKS WHATSOEVER ARE PERFORMED ON THE CONTENTS OF THE TRANSFERRED DATA.

PUT DATA

Similar to GET DATA, except that OS will output ICBLEN bytes from the buffer specified by ICBADR . Again, no data checks are performed.

B) Commands Unique to the Disk File Manager System

Figure 8-2 shows several OS/A+ system commands not yet discussed. These "extended" commands are accessed via the extended request routine in a device driver's handler table (see section 8.3 for details on device drivers). However, some of these extended commands as implemented for the disk device in the File Manager System are important enough to deserve their own sections. We'll examine each of the extended disk operations in a little detail:

ERASE, PROTECT, and UNPROTECT

Also known as Delete, Lock, and Unlock, these three commands simply provide OS with a channel number (i.e., the X-register contains IOCB number times 16), a command number (ICCOM), and a filename (via ICBADR). When OS passes control to the FMS, an attempt is made to satisfy the request. Note that the filename may include "wild cards", as in "D:*.??S" (which will affect all files on disk drive one which have an 'S' as the last letter of their filename extension).

RENAME

Very similar to ERASE, et al, in usage. The only difference is in the form of the filename. Proper form is: "[Dn:]oldname.ext,newname.ext" Note that the disk device specifier is not and CAN NOT be given twice.

NOTE and POINT

Other than OPEN, these are the only commands encountered in standard OS/A+ which use any of the AUXilliary bytes of the IOCB. For these commands, the user specifies the channel number and command number and then receives or passes file pointer information via three of the AUX bytes. ICAUX3/ICAUX4 are used as a conventional 6502 LSB/MSB 16-bit integer: they specify the current (NOTE) or the to-be-made-current (POINT) sector within an already OPENed disk file. ICAUX5 is similarly the current (NOTE) or to-be-made-current (POINT) byte within that sector. In the case of OS/A+ version 4, the word "sector" might be more properly replaced with the word "page", since the NOTE/POINT addressing always uses 256 byte pages, regardless of the physical sector size.

FMS Extensions of the OPEN Command

Open is not truly an extended operation, but for disk I/O we need to know that the FMS allows two additional "modes" beyond the fundamental OS modes.

If ICAUX1 contains a 6 when OS/A+ is called for OPEN, then the disk DIRECTORY is opened (instead of a file) for read- only access. The address ICBADR now specifies the file (or files, if wild cards are used) to be listed as part of a directory listing. Note that FMS expects this type of OPEN to be followed by a succession of GETREC (get text line) OS calls.

If ICAUX1 contains a 9, the specified file is opened as a write-only file, but the file pointer is set to the current end-of-file. CAUTION: version 4 FMS only appends on sector boundaries (it links a new sector to start the append, so there may be used space on the old last sector).

Finally, under version 4 FMS, mode 13 is also legal, specifying that the file be opened in "Append/Update" mode. See Appendix A.1 for more details on the meanings of other bits in ICAUX1 and ICAUX2 under version 4 of OS/A+.

8.1.3 Error Codes Returned

On return from any OS call, the Y-register contains the completion code of the requested operation. A code of one (1) indicates "normal status, everything is okay". (I know, why not zero, which is easier to check for. Remember, we based this on Atari's OS ROMs, which are good, not perfect.) By convention, codes from $Ø2 to $7F (2 through 127 decimal) are presumed to be "warnings". Those from $8Ø to $FF (128 through 255 decimal) are "hard" errors. These choices facilitate the following assembly language sequence:

```
JSR CIOV  ; call the OS
TYA               ; check error code
BMI OOPS  ; if $8Ø-$FF, it must be an error
```

In theory, OS/A+ always returns to the user with condition codes set such that the TYA is unnecessary. In practice, that's probably true; but a little paranoia often leads to longer life of both humans and programs.

8.2: Manipulation of OS/A+

The writer of assembly language code will most likely need to interface with the Atari Operating System (OS) in some way. If the assembly code is to become an extrinsic command, there may be a need to interface to OS/A+. See section 8.2 for further information about the OS interface.

If you are writing software designed to interface with OS/A+, you may need to examine and/or modify certain special memory locations or access certain routines within OS/A+. This section lists and describes those that we feel are the most useful.

8.2.1 SYSEQU.ASM

Every OS/A+ master disk contains an assembler source file, SYSEQU.ASM, that has various commonly used Atari OS and OS/A+ system equates. This file may be included in an assembly language program via the OSS MAC/65 include function (.INCLUDE #D1:SYSEQU.ASM); however, it exists on the master disk as a text file and must be 'ENTER'ed into MAC/65 and then 'SAVE'ed back to the disk.

8.2.2 OS/A+ MEMORY LOCATION

OS/A+ on the Atari is designed to be placed just after the Atari File Manager. Since the actual location of OS/A+ may vary with different versions of a file manager, a fixed location has been assigned to point to OS/A+. The location CPALOC($ØA on the Atari) contains the address of the OS/A+ warmstart entry point. Most Atari programs should return to OS/A+ by JMPing to the address contained in CPALOC.

8.2.3 EXECUTE PARAMETERS

The OS/A+ execute flag is located CPEXFL ($ØB) from the start of OS/A+. The CPALOC may be used as an indirect pointer to access the execute flag:

```
LDY     #CPEXFL          ;GET DISPL TO FLAG
LDA     (CPALOC),Y       ;LOAD FLAG
```

The Execute Flag has four bits that control the execute process:

Name	Bit #	
EXCYES	$8Ø	If one, an execute is in progress
EXCSCR	$4Ø	If one, do not echo execute input to screen
EXCSUP	$2Ø	If one, a cold start execute is starting.Used to avoid a FILE NOT FOUND error if STARTUP.EXC is not on boot disk.
EXCNEW	$1Ø	If one, a new execute is starting. Tells OS/A+ to start with the first line of the file

OS/A+ performs the execute function by OPENing the file, POINTing to the next line, READing that line, NOTEing the new next line and CLOSEing the file. To perform these functions, OS/A+ must save the execute file name and the three byte NOTE values. The filename is saved at CPEXFN ($ØC) into OS/A+. The three NOTE values are saved at CPEXNP($1C) into OS/A+. (CPEXNP=ICAUX5; CPEXNP+1=ICAUX4; CPEXNP+2=ICAUX3). By changing the various execute control parameters, a programmer can cause chaining of execute files.

8.2.4 DEFAULT DRIVE LOCATION

Under Atari version 2, the OS/A+ default drive file spec is located at CPDFDV ($Ø7) into OS/A+. The Default Drive here is ATASCII Dn: where "n" is the ATASCII default drive number.

8.2.5 EXTRINSIC PARAMETERS

The extrinsic commands may be called with parameters typed on the command line. The OSS command

```
D1:COPY  FROMFILE  D2:TO FILE
```

is an example of this. The entire command line is saved in the OS/A+ input buffer located at CPCMDB ($4Ø) bytes into OS/A+ and is available to the user. Since most command parameters are file names, OS/A+ provides a means of extracting these parameters as filenames. The routine that performs this service begins at CPGNFN ($Ø3) bytes into OS/A+ . The routine will get the next parameter and move it to the filename buffer at CPFNAM ($21) bytes in OS/A+. If the parameter does not contain a device prefix, then OS/A+ will prefix the parameter with the default drive prefix. The first time COPY calls CPGNFN the file spec "D1:FROMFILE" is placed at CPFNAM. The second time COPY calls CPGNFN the file spec "D2:TO FILE" is placed in CPFNAM. If CPGNFN were to be called more times, then the default file spec would be set into CPFNAM at each call. To detect the end of parameter condition, the user may check the CPBUFP ($ØA into OS/A+) cell. If CPBUFP does not change often a CPGNFN call then there are no more parameters. The filename buffer is always padded to 16 bytes with ATASCII EOL ($9B) characters. The following example sets up a vector for calling the get file name routine:

```
        CLC
        LDA     CPALOC          ;ADD CPGNFN
        ADC     #CPGNFN         ;TO CPALOC VALUE
        STA     GETFN+1         ;AND PLACE IN
        LDA     CPALOC+1        ;ADDRESS FIELD
        ADC     #Ø              ;OF JUMP
        STA     GETFN+2         ;INSTRUCTION
GETFN   JMP     Ø
```

The following routine gets the next file name to CPFNAM:

```
        LDY     #CPBUFP         ;SAVE CPBUFP
        LDA     (CPALOC),Y      ;VALUE
        PHA
        JSR     GETFN           ;GET NEXT FILE PARM
        LDY     #CPBUFP
        PLA                     ;TEST FOR NO NEXT
        CMP     (CPALOC),Y      ;PARM
        BEQ     NONEXT          ;BR IF NO NEXTPARM
        LDY     #CPFNAM         ;ELSE GET FILE
        LDA     (CPALOC),Y      ;NAME FROM BUFFER
```

8.2.6 RUNLOC

Whenever an Extrinsic command is invoked, RUNLOC ($3D into OS/A+) is given the value of the first address in that command's .COM file. Some Extrinsic commands (including user written commands) can therefore be restarted by typing the RUN command. You may want to change the contents of RUNLOC to point to the warmstart point ofyour program when it's entered the first time to avoid unwanted reinitializations when re-entered. BASIC A+ and MAC/65 do this to avoid clearing any user program which may be in memory when returning from OS/A+. If you want to forbid re-entry, you need to set RUNLOC's high order byte ($3E into OS/A+) to zero:

```
LDY     #RUNLOC+1       ;FORBID RE-ENTRY
LDA     #Ø              ;TO ME
STA     (CPALOC),Y
```

8.3: DEVICE HANDLERS

As we have noted before, CIO is actually a very small program (approximately 7ØØ bytes). Even so, it is able to handle the wide variety of I/O requests detailed in the first two parts of this chapter with a surprisingly simple and consistent assembly language interface. Perhaps even more amazing is the purity and simplicity of the OS interface to its device handlers.

Admittedly, because of this very simplicity, OS/A+ is sometimes slower that one would wish (only noticeably so with PUT BINARY RECORD and GET BINARY RECORD) and the handlers must be relatively sophisticated. But not too much so, as we will show.

8.3.1 The Device Handler Table

At location "HATABS" in RAM, OS/A+ has (loaded from ROM on the Atari) a list of the standard devices (P:, D:,E:,S:, and K:) and the addresses thereof. To add a device, simply tack it on to the end of the list: you need only specify the device's name (one character) and the address of its handler table (more on that in a moment).

In theory, all named device handlers under OS/A+ may handle more than one physical device. Just as the disk handler understands "D1:" and "D2:", so could a keyboard handler understand "K1:" and "K2:". OS/A+ supplies a default sub-device number of "1" if no number is given (thus "D:" becomes "D1:").

Following is the layout of the HAndler TABleS on the Atari:

```
      *=      $Ø31A
   HATABS
      .BYTE   'P'      ; the Printer device
      .WORD   PDEVICE  ; and the address of its driver
      .BYTE   'C'      ; the Cassette device
      .WORD   CDEVICE
      .BYTE   'E'      ; the screen Editor device
      .WORD   EDEVICE
      .BYTE   'S'      ; the graphics Screen device
      .WORD   SDEVICE
      .BYTE   'K'      ; the Keyboard device
      .WORD   KDEVICE
      .BYTE   Ø        ; zero marks the end of the
                         table
      .WORD   Ø        ; ...but there's room for
                       several
      .BYTE   Ø        ; ...more devices
                       et cetera
```

8.3.2 Rules for Writing Device Handlers

Each device which has its handler address placed into the handler address table (above) is expected to conform to certain rules. In particular, the driver is expected to provide six (6) action subroutines and an initialization routine. (In practice, the current OS/A+ only calls the initialization routines for its own pre-defined devices. Since this may change in the future, and since one can force the call to one's own initialization routine, we must recommend that each driver include one, even if it does nothing.) The address placed in the handler address table must point to, again, another table, the form of which is shown below (Figure 8.1).

```
   HANDLER
      .WORD   <address of OPEN routine>-1
      .WORD   <address of CLOSE routine>-1
      .WORD   <address of GETBYTE routine>-1
      .WORD   <address of PUTBYTE routine>-1
      .WORD   <address of STATUS routine>-1
      .WORD   <address of XIO routine>-1
      JMP     <address of initialization routine>
```

Figure 8-3

Notice the six addresses which must be specified; and note that in the table one must subtract one from each address (the "-1" simply makes CIO's job easier...honest). A brief word about each routine is given in the following pages.

Device OPEN

The OPEN routine must perform any initialization needed by the device. For many devices, such as a printer, this may consist of simply checking the device status to insure that it is actually present. Since the X-register, on entry to each of these routines, contains the IOCB number being used for this call, the driver may examine ICAUX1 (via LDA ICAUX1,X) and/or ICAUX2 to determine the kind of OPEN being requested. (Caution: OS/A+ preempts bits 2 and 3 ($Ø4 and $Ø8) of ICAUX1 for read/write access control. These bits may be examined but should normally not be changed.)

Device CLOSE

The CLOSE routine is often even simpler. It should "turn off" the device if necessary and possible.

Device PUT and GET BYTE Routines

The PUTBYTE and GETBYTE routines are just what are implied by their names: the device handler must supply a routine to output one byte to the device and a routine to input one byte from the device. HOWEVER, for many devices one or the other of these routines doesn't make sense (ever tried to input from a printer?). In this case the routine may simply RTS and OS/A+ will supply an error code.

Device STATUS Routine

The STATUS routine is intended to implement a dynamic status check. Generally, if dynamic checking is not desirable or feasible, the routine may simply return the status value it finds in the user's IOCB. However, it is NOT an error under OS/A+ to call the status routine for an unOPENed device, so be careful.

Device Extended I/O Routine(s)

The XIO routine does just what its name implies: it allows the user to call any and all special and wonderful routines that a given device handler may choose to implement. OS does nothing to process an XIO call except pass it to the appropriate driver.

General Comments on Device I/O Routines

In general, the AUXilliary bytes of each IOCB are available to each driver. In practice, it is best to avoid ICAUX1 and ICAUX2, as several BASIC and OS commands will alter them to their will. Note that ICAUX3 thru ICAUX5 may be used to pass and receive information to and from BASIC via the NOTE and POINT commands (which are actually special XIO commands). Finally, drivers should not touch any other bytes in the IOCBs, especially the first two bytes.

Notice that handlers need not be concerned with PUT BINARY RECORD, GET TEXT RECORD, etc.: OS performs all the needed housekeeping for these user-level commands.

8.3.3 Rules for Adding Things to OS
---

1. Inspect the system MEMLO pointer (see SYSEQU.ASM for the actual location).
2. Load your routine (including needed buffers) at the current value of MEMLO.
3. Add the size of your routine to MEMLO.
4. Store the resultant value back in MEMLO.
5. Connect your driver to OS by adding its name and address into the handler address table.
6. For Atari handlers only:
   Fool OS so that if SYSTEM RESET is hit steps 3 thru 5 will be reexecuted (because SYSTEM RESET indeed resets the handler address table and the value of MEMLO).

In point of fact, step 2 is the hardest of these to accomplish. In order to load your routine at wherever MEMLO may be pointing, you need a relocatable (or self-relocatable) routine. Since there is currently no assembler for OS/A+ which produces relocatable code, this is not an easy task. But it may not be necessary if you are writing code for your own private system instead of the general public.

Step 6 is accomplished by making Atari OS think that your driver is the Disk driver for initialization purposes (by "stealing" the DOSINI vector) and then calling the Disk's initializer yourself before steps 3 thru 5 are performed again.

8.3.4 AN EXAMPLE PROGRAM

This driver, included in source form on your disk as "MEM.LIS", builds a new driver and adds it to the operating system. The "device" being driven is simply excess system memory within your computer. Thus, you may (for example) use this are as a pseudo-disk file for passing data between sequentially called programs.

Some words of caution are in order. This driver does NOT perform step 6 as noted in the last section (but it may be reinitialized via a BASIC USR call). It does NOT perform self-relocation: instead it simply locates itself above all normal low memory usage (except the serial port drivers, which would have to be loaded AFTER this driver). If you assemble it yourself, you could do so at the MEMLO you find in your normal system configuration (or you could improve it to be self-modifying, of course).

Other caveats pertain to the handler's usage: it uses RAM from the contents of MEMTOP downward. It does NOT check to see if it has bumped into BASIC's MEMTOP ($90) and hence could conceivably wipe out programs and/or data. To be safe, don't write more data to the RAM than a FRE(0) shows (and preferrably even less).

In operation, the M: driver reinitializes upon an OPEN for write access (mode 8). A CLOSE followed by a subsequent READ access will allow the data to be read in the order it was written. MORE CAUTIONS: don't change graphics modes between writing and reading if the change would use more memory (to be safe, simply don't change at all). The M: will perform almost exactly as if it were a cassette file, so the user program should be data sensitive if necessary: the M: driver will NOT itself give an error based on data contents. Note that the data may be re-READ if desired (via CLOSE and re-OPEN).

A suggested set of BASIC programs is presented on the next page.

```
Ending of PROGRAM 1:
       9900 OPEN #2,8,0,"M:"
       9910 PRINT #2; LEN(A$)
       9920 PRINT #2; A$
       9930 CLOSE #2
       9940 RUN "D:PROGRAM2"

Beginning of PROGRAM 2:
       100 OPEN #4,4,0,"M:"
       110 INPUT #4,SIZE
       120 DIM STRING$(SIZE)
       130 INPUT #4, STRING$
       140 CLOSE #4
```

BASIC A+ users might find RPUT/RGET and BPUT/BGET to be useful tools here instead of PRINT and INPUT. And, of course, users of any other language(s) might find this a handy inter-program communications device.

CHAPTER 9: FILE STRUCTURE

9.1: Version 2 File Structure

OS/A+ version 2 was produced to provide the maximum compatibility possible with Atari's DOS 2.Øs. In fact, the FMS used is identical to that used by Atari (for a simple reason: we wrote Atari's DOS). For reasons known best to Atari, we were instructed to create Atari's FMS around a linked-sector disk space management scheme. In essence, this means that the last three bytes of each sector in a disk file contain a link to the next sector in that same file. The positive result of this is that one produces a relatively small, memory-resident, disk manager which is nevertheless capable of dynamically allocating diskette space (unlike, for example, a contiguous file disk manager). The biggest disadvantage of the scheme seems to be that one may not do direct (random) access to the bytes of such files, as one CAN do with either a contiguous or mapped file allocation technique. Also, a disk error in the middle of a linked file means a loss of access to the rest of the file.

The purpose of the FMS is to organize the 72Ø data sectors avilable on an 81Ø (or its double density equivalent) diskette into a system of named data files. FMS has three primary data structures that it uses to organize the disk:

1. Volume Table of Contents (VTOC): a single disk sector which keeps track of which disk sectors are available for use in data files.

2. Directory: a group of eight contiguous sectors used to associate file names with the location of the files' sectors on the disk. Each Directory entry contains a file name, a pointer to the first data sector in the file, and some miscellaneous information.

3. Data Sectors: sectors containing the actual data and some control information that links one data sector to the next data sector in the file.

NOTE: since double density diskette sectors contain 256 bytes whereas single density (81Ø drive) sectors contain only 128, certain absolute byte number references may vary depending upon the diskette in use. Throughout this chapter, in such cases, the single density number is given followed by the double density number in square brackets [thus].

9.1.1 DATA SECTORS

A Data Sector is used to contain the file's data bytes. Each 128 [256] byte data sector is organized to hold 125 [253] bytes of data and three bytes of control. The data bytes start with the first byte (byte Ø) in the sector and run contiguously up to, and including, byte 124 [252]. The control information starts at byte 125 [253].

The sector byte count is contained in byte 127 [255]. This value is the actual number of data bytes in this particular sector. The value may range from zero (no data) to 125 [253] (a full sector). Any data sector in a file may be a short sector (contain less than 125 [253] data bytes).

The left six bits of byte 125 [253] contain the file number of the file. This number correspoinds to the location of the file's entry in the Directory. Directory entry zero in Directory sector $169 has a file number of zero. Entry one in Directory sector $169 has a file number one, and so forth. The file number value may range from zero to 63 ($3F). The file number is used to insure that the sectors of one file do not get mixed up with the sectors of another file.

The right two bits of byte 125 [253] (and all eight bits of byte 126 [254]) are used to point to the next data sector in the file. The ten bit number contains the actual disk sector number of the next sector. Its value ranges from zero to 719 ($2CF). If the value is zero then there are no more sectors in the file sector chain. The last sector in the file sector chain is the End-Of-File sector. The End-Of-File sector will almost always be a short sector.

9.1.2 DISK DIRECTORY

The Directory starts at disk sector $169 and continues for eight contiguous sectors, ending with sector $17Ø. These sectors were chosen for the directory because they are in the center of the disk and therefore have the minimum average seek time from any place else on the disk. Each directory sector has space for eight file entries. Thus, it is possible to have up to 64 files on one disk.

A Directory entry is 16 bytes in size, as illustrated by Figure 9-2. The directory entry flag field gives specific status information about the current entry. The directory count field is used to store the number of sectors currently used by the file. The last eleven bytes of the entry are the actual file name. The primary name is left justified in the primary name field. The name extension is left justified in the extension field. Unused filename characters are blanks ($20). The Start Sector Number field points to the first sector of the data file.

Starting Byte # of Field	Length of Field (bytes)	Purpose of Field
Ø	1	Flag byte. Meanings of bits: $ØØ Entry never used $8Ø Entry was deleted $4Ø Entry in use $2Ø Entry protected $Ø2 a version 2 file $Ø1 Now writing file
1	2	Count (LSB,MSB) of sectors in file
3	2	Start sector (LSB,MSB) of link chain
5	8	File name, primary
13	3	File name, extension

Figure 9-2

Directory Entry Structure

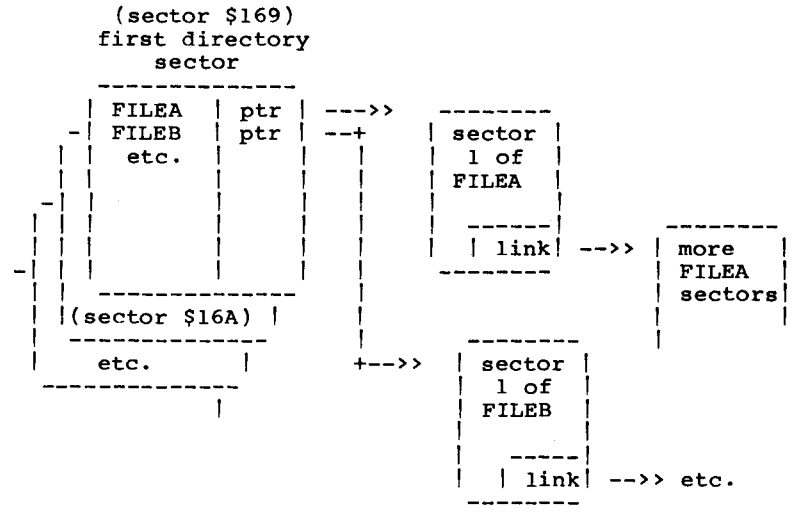

Figure 9-2

Version 2 Directory Structure

NOTE: only eight file directory entries are stored per sector, even on double density diskettes.

9.1.3 VOLUME TABLE OF CONTENTS (VTOC)

The VTOC sector ($168) is used to keep track of which disk sectors are available for data file usage. Figure 9-3 illustrates the organization of the VTOC sector. The most important part of the VTOC is the sector bit map.

The sector bit map is a contiguous string of 90 bytes, each of which contains eight bits. There are a total of 720 (90 x 8) bits in the bit map--one for each possible sector on an 810 diskette. The 90 bytes of bit map start at VTOC byte ten ($0A). The leftmost bit ($80 bit) of byte $0A respresents sector zero. The bit just to the right of the leftmost bit ($40 bit) represents sector one. The rightmost bit (bit $01) of byte $63 represents sector 719.

Starting Byte # of Field	Length of Field (bytes)	Purpose of Field
0	1	Reserved (for type code)
1	2	Total number of sectors
3	2	Number of unused sectors
5	5	Reserved
10	90	Sector usage bit map Each bit represents a particular sector: a 1 bit indicates an available sector, a 0 bit indicates a sector in use.
100	28	Reserved (could be used for version 2 type DOS with more than 720 sectors per disk)

Figure 9-3

Structure of the VTOC Sector

9.2 Version 4 File Structure

OS/A+ version 4 is an operating system which provides all the power and flexibility of the Atari CIO scheme, and also uses an advanced File Management System (FMS) to provide fast and effective random access files. OS/A+ version 4, as it appears to the user, is virtually identical with OS/A+ version 2, except that it work with disk drives ranging in storage size from 128K bytes to over 15 Megabytes.

OS/A+ version 4 utilizes a mapped file structure which allows true random access to data files. In such a scheme, special segments of a file act as pointers to the blocks of data comprising a file. By allowing the user to specify the size of the data blocks pointed to, OS/A+ is able to handle large and small files and disks with unsurpasssed speed and utility.

The OS/A+ random access file managment system treats each disk under its control as a collection of contiguous physical sectors of either 256 or 512 bytes in length, which are numbered starting with sector zero. These sectors are logicaly grouped into blocks of n sectors in length, where n is a power of two between 1 and 128. All files on a disk are allocated space in segments of at least one block in length.

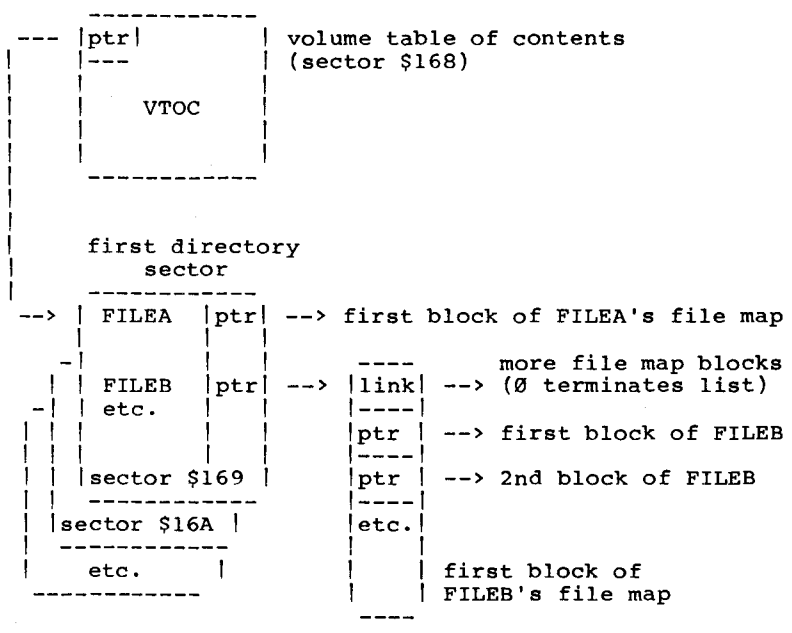

Figure 9-4

Version 4 File Structure

There are several non-obvious advantages to this scheme, so bear with us as we try to explain some of them.

A. We are able to handle disks with 128, 256, or 512 bytes per sector. (To be truthful, with 128 bytes per sector drives we would use pairs of sectors to emulate 256 byte sectors, since a 128 byte file map is not really adequate.)

B. We allow each DISK DRIVE to be assigned its own "drive blocking factor". That means that a quadruple density floppy might have blocks consisting of two 256-byte sectors while a 1Ø MB disk might use blocks of four 512-byte sectors. Note that this concept of blocking factors is not new or unique: CP/M 2.2 allows blocking factors of 1KB, 2KB, and 4KB, depending on disk size. We are simply a little more flexible.

C. We allow each FILE to be assigned its own "file blocking factor". Thus, even on a floppy with 512 byte blocks, a given FILE may use 8 KByte blocks, thus guaranteeing at most one disk read to access any given sector of the file. (On the Apple II version of this product, where the drive blocking factor is perforce 1 for standard Apple drives, a file blocking factor of 8 -- 2 KByte blocks -- essentially doubles random access speed.)

D. Although not yet implemented nor planned for first release, the directory structure is set up in such a way that, if desired, we could implement multiple and/or hierarchical directories (ala UNIX, for example). Even CIO (on the Apple II) has been altered to support the concept of a default device and/or directory.

E. Random access files are easy and practical. Unix-like "LSEEKs" are accomplished (via the "POINT" XIO call in section x.xx) to any byte of any file.

F. Except for those rare programs that somehow depend on having 125 bytes of data per sector, current Atari application programs (including Atari BASIC and programs written thereunder) will notice NO CHANGE in their interface with the operating system. Of course, some of the currently unused options will be available to take advantage of such features as file blocking factors, but they will not be necessary to the proper functioning of the system.

9.2.1 THE VERSION 4 VTOC

In order to keep track of what blocks on a partiular disk are available for use,the file manager maintains a special section on each disk known as the volume table of contents, or VTOC. The VTOC on a disk consists of 1 or more sectors which contain the bitmap (a long string of bits in which the Nth bit represents the Nth block on the disk). Each bit may be turned on or off to free or allocate (respectively) the block it represents. Note that the VTOC is the only data area on the disk allocated by sectors instead of blocks. The format of the VTOC is as follows:

hex offset	value
Ø	unused
1-2	block no. of first directory sector
3	unused
4-5	unused
6	unused
7-26	unused
27	max. number of pointers in file map sectors ($7A for disks with 256-byte sectors; $F4 for 512-byte sector disks)
28-2F	unused
3Ø-33	unused
34	unused
35	unused
36-37	unused
38-	disk block bit map

9.2.2 THE DIRECTORY

The disk directory holds information describing the existing files on a particular disk. The directory is allocated on the disk by disk blocks, each sector of which holds information on a certain number of files (7 for 256-byte sectors, 14 for 512-byte sectors). The blocks themselves are linked, each one holding the disk address (block number) of its successor, with the last block having a null link (block number = Ø). Each entry of the directory describes a particular file. An entry for a single file contains the file's name, its length in sectors (see exceptions for DOS 3.3 diskettes), its file type byte (see below), and a pointer to the start of the file map for that file.

The file name consists of a string of up to 3Ø characters excluding spaces, commas, carriage returns, or nulls. Characters within a directory entry will have their upper bit inverted. the file name will be padded at the right with inverse video blanks (hex $AØ).

The file type byte is used as follows:

BITS Ø-3: file use type -- values Ø,1,2, and 4 are currently used on the Apple system while only Ø is used on the Atari.

BITS 4-6: file blocking factor -- a value from Ø to 7 (see next section)

BIT 7: file protection -- if this bit is set then the file is protected from accidental write access, erasure, or renaming. The pointer to the start of the file map is a two byte value which is the disk block number of the first file map map block.

Format of directory sectors:

hex offset	value
Ø	unused
1-2	block number of next directory block (Ø if this is the last block)
3-A	unused
B-	directory entries (35 bytes each)

Format of directory entries:

hex offset	value
Ø-1	block number of first block in file map
2	file type
3-2Ø	file name
21-22	length of file in sectors

9.2.3 THE FILE MAP

As previously mentioned OS/A+ version 4 utilizes a mapped file structure where special portions of a file point to the locations on the disk holding the actual data. These special sections comprise the file map, which is a singly linked list of disk blocks which contain pointers to the data blocks of a file. Each file map sector has the following format:

hex offset	value
Ø	unused
1-2	block number of next file map block (zero if this block is the last)
3-4	unused
5-6	unused
7-B	unused
C-	sequential list of block numbers in file (2 bytes per block number)

A pointer to a file block is merely the disk block number of the start of a file block. For most purposes, a file block is equivalent in size to a disk block, However, the file manager allows the user to specify a file blocking factor which alters the size of data blocks for a single file. A file blocking factor of zero implies that file blocks are equivalent to disk blocks. A file blocking factor of 1, though, makes file blocks equivalent to 8 disk blocks in length. Similarly, a factor of 2 creates file blocks which are 16 disk blocks in length. While the use of a file blocking factor has little or no consequence for sequentially accessed files, it offers 2 distinct advantages for random access files. First, the size of the file map will be reduced, thereby decreasing the average number of disk accesses requrired to access data. Second, the file's data sectors are likely to be less fragmented on the disk, thereby decresing the average head movement required to access data sectors. The only disadvantage of using a file blocking factor is that disk space is allocated much more rapidly than otherwise, making this technique undesirable for small files.

9.2.4 BUFFER ALLOCATION

The file manager requires a continuous block of memory from which to allocate buffer space. Data passing between disk files and user programs is temporarily stored here. The address of the start of this space is contained in location SASA (see system memory map), and its length in pages (256 bytes) is in SABYTE. The values in these locations may be changed by the user whenever all files are closed; however, the system must then be re-initialized either by jumping to the system reset location or by calling the file manager initialization subroutine.

The buffer space is allocated in this manner:

1. When the system is booted (or reinitialized), space for each disk drive's VTOC is allocated. The space required for a given VTOC is its length in bytes.

2. When a file is opened on a given disk drive, space is required to hold 2 of that drive's sectors.

EXAMPLE: Suppose a system has 2 disks, both with 256-byte sectors. There will be 2 VTOC buffers (a VTOC is usually 1 sector long), so there's a total of 512 bytes. Each open file will have file map and data buffers, a total of 512 bytes per open file (two 256-byte sectors). Therefore, allowing for 3 open files at one time, there should be three 512-byte buffers for the files, plus the 512 bytes of buffer for the VTOCs, for a total of 8192 bytes (8 pages) of buffer space (i.e., SABYTE = 8).

9.2.5 ADDING DRIVES

In order to integrate a new disk drive into the system, the disk must first be initialized with the OS/A+ verion 4 file structure. Please refer to the section describing the INIT utility for instructions on using INIT. Source code of INIT is available for users contemplating adding their own disk devices.

In order to access a new drive, it must be installed into the disk drive table within the file manager. This table consists of 8 entries of 16 bytes each starting at location DRVTAB (see memory map). Each entry in this table contains the folowing information:

byte(s)	value	
Ø	drive type--	1=Apple Disk II Ø=other disk
1	reserved	
2	size of disk sectors--	1=256 byte 2=512 byte
3	disk blocking factor; sectors/block	
4-5	sector no. of disk's VTOC	
6	no. of sectors in VTOC	
7-8	address of read/write sector routine minus one	
9-1Ø	(set by file manager)	
11	file map sector size-- $7A for 256 byte sectors; $F4 for 512 byte sectors	
12-15	(set by file manager)	

As can be seen, the drive table entry contains the address of the routine to read and write sectors on the disk. This routine may be located anywhere in memory. (See the next section for a description of the parameters to this routine).

Once the drive has been added to the drive table and the read write sector routine has been loaded, the system must be re-initialized by jumping to the system reset location. The new disk may then be accessed as Dn: where the disk is the nth entry in the drive table (n starts from 1).

NOTE: more than one disk drive may be serviced by a single read/write sector routine.

9.2.6 THE READ/WRITE SECTOR ROUTINES

Each read/write sector routine receives its parameters through the Device Control Block, or DCB (see memory map). The format of the DCB is as follows:

Byte Ø:	machine dependent (unused on Apple)
Byte 1:	DCBDRV disk drive number (1-8).
Byte 2:	DCBCMD command (Ø=null, 1=read, 2=write).
Byte 3:	DCBSTA status byte (set by read/write sector).
Bytes 4-5:	DCBBUF buffer address in low, high order.
Bytes 6-7:	machine dependent (unused on Apple)
Bytes 8-9:	machine dependent (unused on Apple)
Bytes 9-A:	DCBSEC sector number--to be accessed in low, high order

See the table in Chapter 13 for valid status values.

CHAPTER 10: VERSION DIFFERENCES

As much as we would like to, we can't produce all three versions of OS/A+ in such a manner that they will be 100% compatible. And, of course, there are also minor differences between OS/A+ and Atari DOS.

In this chapter we will try to document as many known major differences as we can. We will probably miss one or two, so please don't hesitate to call or write us if you intend to write software which will run on more than one version of OS/A+. We will try to keep a list of any other differences we (or you) find.

Of course, if you are working entirely within one of our OS/A+ configurations, most of this is unneeded information. Perhaps you need to be aware only of the differences between OS/A+ and your computer's "normal" operating system.

10.1 FEATURES SPECIFIC TO VERSION 4 OF THE FILE MANAGER

OS/A+ version 4 contains several capabilities and enhancements which are not available under version 2 or Atari DOS. A summary of these added features follows:

10.1.1 RANDOM ACCESS:

As previously mentioned, OS/A+ version 4 allows true random access to data files. This capability is provided through the usage of the NOTE and POINT operating system calls. Unlike its predecessor, version 4 expects the data in ICAUX3-ICAUX5 to be a 24 bit RELATIVE position within a file (in mid, high,low byte order). This allows the user to use the POINT command to easily move to any position within a file of 16 megabytes or less in size. Similarly, the data returned by the NOTE command is a 3 byte RELATIVE position value which is placed in ICAUX3-ICAUX5 by the file manager. Notice also that a random access file may contain "holes" created by writing non-contiguous portions of a file without writing the intervening data.

10.1.2 FILE TYPES:

Version 4 of OS/A+ also supports a file type byte which is also used to indicate the file's file map blocking factor (see section 10.4) and its protection. When a file is created, this byte is set into the file's directory entry; its value is zero by default. In order to place a value other than zero into the file type byte when a file is created, bit 6 of ICAUX1 must be set (i.e. add $40, or 64, to the mode during an OPEN command). The desired file type must be placed in ICAUX2 (the second value in a BASIC open statement). Whenever a file is opened, the current value of its type byte is returned in ICAUX2.

10.1.3 SUPPRESSING AUTOMATIC FILE CREATION:

Normally, when a file is opened in mode 8, it is created, if the file did not previously exist, or truncated, if it did exist. It is occasionally desirable to suppress creation or truncation when opening a file in mode 8, in which case an error would occur when attempting to open a non-existent file. If a file is opened in mode 8 and bit 5 of ICAUX1 is set (i.e., add $20, or 32, to the mode value), then the creation or truncation of the file will be prevented.

10.2 DIFFERENCES: ATARI DOS AND OS/A+

There are very few points of difference between Atari DOS and version 2 of OS/A+, other than the fact that Atari DOS uses DUP.SYS while OS/A+ uses its Command Processor. And, actually, there are few differences between version 2 and version 4 of OS/A+ AS SEEN BY AN APPLICATIONS PROGRAM (including BASIC A+, etc.). Since the differences are generic, rather than specific to a particular version, they will be discussed by category.

10.2.1 MEMORY USAGE

The only real problem that can exist here is in the location of LOMEM, the beginning of user application memory. If a program written for Atari DOS (or OS/A+ version 2, for that matter) has assumed a particular LOMEM, it may not run under a version with a higher LOMEM. To illustrate, the following table lists the LOMEM value that will result in each of several cases if we assume a system configuration of 2 disk drives allowing 3 disk files open at the same time.

version	LOMEM contents
Atari DOS 2.0s, single density disks	$1C00
Atari DOS 2.0s, double density disks	$1E80
OS/A+ version 2, single density disks	$1F00
OS/A+ version 2, double density disks	$2180
OS/A+ version 4	$2C00

Of course, by changing the contents of SASA and SABYTE (see chapters 12 and 13), the user may change the location and number of buffers, so a certain measure of LOMEM independence may be obtained by, for example, placing the buffers somewhere within an application program's memory space. However, even this is not foolproof: examine the memory map of chapter 13 for more details.

10.2.2 END OF FILE

Atari DOS and OS/A+ version 2 both are capable of knowing exactly where a file ends, since each sector "knows" (via its 3 byte link information) how many bytes it contains. OS/A+ version 4, however, DOES NOT KNOW exactly where the END OF FILE is. In fact, version 4 will not report end of file until it reaches the end of the last sector in the file. Normally, this has little if any effect on a user program.

In particular, if reading text lines, the system will read the last line the same on either version. Then, when an attempt is made to read the first line past the end of file, version 2 reports an immediate error. Version 4, however, will begin passing the trailing zero bytes to CIO. However, CIO will not end the transfer until it receives an ATASCII RETURN code ($9B) (it expects to return an error 137, truncated record, if the user buffer is too short for all those null bytes). But the file manager finally returns the end of file signal, and CIO ignores any previous errors to return the EOF code.

With binary files, the problem is more subtle, since a binary record of all nulls is perfectly legal. However, most user programs on the Atari will have been built in such a way as to be aware of how many records are in a given file. And, if they have not been so built, there is usually at least one field in the record which cannot contain a null result. That field may be checked for nulls as an end of file test.

NOTE: virtually any Atari DOS program which used indexed files (via NOTE and POINT) will function properly under version 4. Of course, programs which "take over" the entire disk may fail, but that is because there are 256 bytes per sector and their direct SIO calls are now improper, which has nothing to do with DOS.

10.2.3 RANDOM ACCESS

This subject has been treated more thoroughly in preceding sections and pages, but let us at least mention here that programs using NOTE and POINT properly under DOS 2.0 or OS/A+ version 2 will need no changes to move to version 4.

Of course, programs using the direct random access capabilities of version 4 (i.e., the ability to POINT relative to the beginning of a file, without the need to have previously NOTEd) will not transfer back to version 2. Sorry, but that's the price one must pay for using advanced features.

10.2.4 FILE BUFFERS

In order for OS/A+ to function properly, it requires special segments of memory (called file buffers) to be allocated for the purpose of temporarily holding information passing between disk files and a user program. For some purposes, the user may wish to move these buffers from their default locations (see system memory map). All OS/A+ systems use the concept of a "Begin Buffer Allocation Here" address and a location telling OS/A+ to "Allocate This Many Buffers". The label SASA in the memory map (or, better, SYSEQU.ASM) defines a word containing the starting address of the buffers. The label SABYTE defines the number of buffers to be allocated.

CAUTION: under Atari DOS and version 2 of OS/A+, SABYTE refers to the number of 128 byte buffers to allocate. Under version 4 of OS/A+, SABYTE contains the number of 256 byte PAGE buffers to allocate.

SECONDARY CAUTION: Remember the allocation requirements differ between single and double density disks under version 2 and Atari DOS.

10.2.5 SECTORS 1, 2, and 3

To insure compatibility with the Atari Computer boot process, OS/A+ (both versions) always reads and writes Sectors 1, 2, and 3 in single density (128 byte) mode. This single density "force" occurs at the BSIO level, so programs using BSIO may do so in a compatible fashion.

APPENDIX A - CUSTOMIZING OS/A+

Although OS/A+ was designed and implemented with the average user in mind, no one piece of software can ever be all things to all people. For that reason, a degree of flexibility exists over certain aspects of the system which allows the user to modify OS/A+ to suit his own tastes. The following sections describe the most useful modifications which may be performed.

A.1 BUFFER ALLOCATION

Both versions of OS/A+ allow the user to specify the starting address of the system file buffers and the number of buffers to be used. The location of the words which specify these parameters varies between version 2 and version 4 and, in any case, is not guaranteed to remain fixed in future releases. Therefore, it is strongly suggested that the user desiring to change one or both of these values check the file "SYSEQU.ASM", supplied on the OS/A+ disk, to be sure of the latest system value. As of the printing of this manual, the following locations are in use:

version	label	location	use
2	SASA	$Ø7ØC	start of buffers
4	SASA	$Ø7ØF	start of buffers
2	SABYTE	$Ø7Ø9	# of buffers
4	SABYTE	$Ø711	# of buffers

Presuming the user wishes to change SABYTE, the first question that needs answered is "How many buffers do I need?" The rules follow:

OS/A+ VERSION 2: For single density diskettes, use 1 buffer per active drive AND 1 buffer per simultaneously open file. For double density diskettes, use 2 buffers per active drive and 2 buffers per simultaneously open file. EACH BUFFER IS 128 BYTES LONG.

OS/A+ VERSION 4: For disks with 256 byte sectors (e.g., floppy disks under double density), use 1 buffer per active drive AND 2 buffers per simultaneously open file. For disks with 512 byte sectors, double both figures. EACH BUFFER IS 256 BYTES LONG.

CAUTION: Note the difference in the size of the buffers specified by the SABYTE contents.

A.2 SPECIFYING EXISTING DRIVES

Under version 4, the only way to specify an existent drive is to add its parameters to the drive table (see Chapter 9). Also, the CONFIGure extrinsic command will configure this drive table for you.

Under version 2, the byte location DRVBYT (at $70A, but consult SYSEQU.ASM to confirm current location) controls which drives are active. Each bit of DRVBYT represents a given drive. The least significant bit of DRVBYT represents drive 1, the next bit represents drive 2, etc., up to the most significant bit which represents drive 8.

If a bit is DRVBYT is on (set to one), the drive is active. If a bit is off, the drive is inactive. Thus a value of $Ø5 would imply that "D1:" and "D3:" are active.

CAUTION: simply changing the bits in DRVBYT or adding information to the disk drive table is NOT sufficient to change the system configuration. After changing the bits, you must cause OS/A+ to reinitialize itself. This may be accomplished by simply hitting the SYSTEM RESET key from the keyboard, or calling the DOS initialization routine, via DOSINI, from a running program.

A.3 SAVING YOUR MODIFIED VERSION

Saving a modified version of OS/A+ is extremely simple. With version 2, simply use the INIT command and, when the menu appears, specify "Write DOS.SYS file only" (or go ahead and initialize the disk if it is a new disk...just be careful not to reinitialize a disk with valuable goodies on it).

With version 4, the process is both more complicated and simpler. Since the version 4 boot process simply searches for the filename "DOS.SYS", any file may be renamed DOS.SYS and the system will attempt to boot it.

Saving a modified OS/A+, then, is as simple as SAVing the proper segment of memory. For OS/A+ version 4 issue the following command from the D1: prompt.

```
SAVE D1:DOS.SYS 7ØØ 24ØØ [RETURN]
```

APPENDIX B - SYSTEM MEMORY MAPS

B.1 - ATARI ZERO PAGE MAP:

location		usage
Ø-9		system zero page
A-B	CPALOC	known to Atari DOS as DOSVEC
C-D	DOSINI	vector to FMS initialization
E-42		system zero page
43-49		fms zero page
4A-7F		system zero page
8Ø-FF		user and language zero page
8Ø-BF		BASIC A+ zero page
D2-FF		floating point zero page

B.2 - ATARI SYSTEM MEMORY MAP-- version 2:

location		usage
1ØØ-1FF		65Ø2 stack area
2ØØ-319		system ram
3ØØ-3ØB		DCB (device control block)
31A-33F		device handler table
34Ø-3BF		IOCB's - 8 at 16 bytes each
3CØ-57F		system ram
58Ø-5FF		E: text buffer
6ØØ-6FF		user ram
7ØØ-1C7F		OS/A+ -- file manager and CP
709	SABYTE	number of 128 byte file buffers
70A	DRVBYT	bit map: accessible drives
70C	SASA	address of start of buffers
1C8Ø-1EFF		file manager buffers-- default size
1FØØ-BFFF		user, language, and graphics memory
CØØØ-FFFF		I/O locations and system rom

B.3 - ATARI SYSTEM MEMORY MAP-- version 4:

location		usage
100-1FF		6502 stack area
200-319		system ram
300-30B		DCB (device control block)
31A-33F		device handler table
340-3BF		IOCB's - 8 at 16 bytes each
3C0-57F		system ram
580-5FF		E: text buffer
600-6FF		user ram
700-1C7F		OS/A+ file manager
70F	SASA	address of start of buffers
712	SABYTE	number of 256 byte buffers
713		disk drive table - 8 entries
1C80-1CFF		read/write sector routine
1D00-23FF		OS/A+ CP -- console processor
2400-2BFF		file manager buffers-- default size
2C00-BFFF		user, language, and graphics memory
C000-FFFF		I/O locations and system rom

APPENDIX C - Errors

C.1 TYPES OF ERRORS

All OS/A+ operations return a status value in the IOSTAT field. OS/A+ convention is that status values of $8Ø or greater indicate some sort of error. There are four fundamental kinds of errors that can occur with OS/A+:

Hardware Errors

Such as attempting to read a bad disk, write a read-only disk, etc.

Data Transfer Errors

Errors which occur when data is transferred between the computer and a peripheral device. Examples include Device Timeout, Device NAK, Framing Error, etc.

Device Driver Errors

Found by the driver for the given device, as in (for the DFM) File Not Found, File Locked, Invalid Drive Number, etc.

OS Errors

Usually fundamental usage problems, such as Bad Channel Number, Bad Command, etc.

ERROR CODE HEX	DECIMAL	MEANING
$Ø1	1	No error or warning.
$Ø2	2	Truncated ASCII line. The OS did not find a CR within BUFLEN for ASCII line I/O.
$Ø3	3	End of file look ahead. The last byte transfered from the device driver was its end-of-file byte. The device driver must set this status, so it is best to verify that the device being used is capable of returning this status before depending on it.
$8Ø	128	Operation aborted. Set by Device Handler. (Also BREAK abort on Atari.)
$81	129	File already open. Program is trying to open a channel (IOCB) that has already been OPENed.
$82	13Ø	Device does not exist. The device was not found in the OS device table. Often caused by forgetting the disk drive name when using a disk file.
$83	131	File is write only. Program tried to read from a file which can only be used for writing (i.e., file was OPENed with AUX1 set to 8 or 9).
$84	132	Invalid Command. CIO has rejected your requested command. (Example: program tried to do XIO to a device which has no extended operations defined.)
$85	133	Device/File not open. The IOCB has not been OPENed for the operation. Most I/O requests require that the channel be OPENed before a request can be made.
$86	134	The IOCB specified is invalid. Only IOCB numbers $ØØ, $1Ø, $2Ø, $3Ø, $4Ø, $5Ø, $6Ø, and $7Ø are valid. From some languages, these will be seen as channels Ø to 7.

$87	135	File is read only. Program tried to write to a file which can only be used for reading (i.e., file was OPENed with AUX1 specified as 4 or 6.
$88	136	End of file. No more data in file.
$89	137	Truncated record error. Usually occurs when the line you are reading is longer then the maximum record size specified in the Call to CIO (line oriented I/O). Can't occur with binary I/O on version 2 OS/A+.
$8A	138	Device timeout error. Usually set by the serial bus I/O handler ("SIO") because a device did not respond within the alloted time as set by the OS.
$8B	139	Device NAK error. Atari: serial I/O error.
$8C	140	Serial framing error. Atari: serial I/O error.
$8D	141	Cursor out of range for specific graphics mode you are in. (Could be used for similar meaning by a non-graphics device.)
$8E	142	Serial bus overflow. Atari: computer could not respond fast enough to serial bus input (SIO error).
$8F	143	Checksum error. Communications over the serial bus are garbled (Atari SIO error).
$90	144	1) Device done error. A valid command on the serial bus was not executed properly. Atari: disk rotational speed needs ad-justment. 2) Write protect error. The diskette has a write protect tab in place.
$91	145	Illegal screen mode error. Bad graphics mode number. Other devices: AUX1 and/or AUX2 bytes in IOCB are illegal.
$92	146	This error means the function you tried to do has not been implemented in the device handler. (Example: attempt to POINT with the graphics device.)

$93	147	Not enough RAM for the graphics mode you requested. (Could be used by custom drivers for a similar message.)

NOTE: Errors $AØ through $AF are file manager errors.

$AØ	16Ø	Either a drive # NOT between 1-8 or drive was not powered on.
$A1	161	Too many OPEN files. No free sector buffers to use for another file.
$A2	162	Disk FULL. No free space left on disk.
$A3	163	Fatal system error. Either DOS has bug or bad diskette.
$A4	164	File mismatch. Bad file structure or POINT values wrong.
$A5	165	Bad file name. Check for illegal characters in file name. Version 4 is more liberal in this regard than version 2.
$A6	166	The byte count in your POINT Call was greater then 125 (for single density version 2) or 253 (for double density version 2).
$A7	167	The file specified is locked (PROtected). Protected files cannot be erased or written to.
$A8	168	The software interface for the specific device received an invalid command (example: tried to access a non-existent track or sector).
$A9	169	All space allocated for the directory has been used up (too many filenames in use).
$AA	17Ø	The file you requested does not appear on this diskette.
$AB	171	You have tried to POINT to a byte in a file that is not OPENed for update (version 2 only).
$AC	172	Tried to OPEN a DOS 1 file with DOS II (version 2 only).
$AD	173	The disk drive has found bad sectors while trying to format the disk.

O S S PRECISION SOFTWARE™
A DIVISION OF O.S.S., INC.
1173D S. Saratoga/Sunnyvale Road San Jose, CA 95129