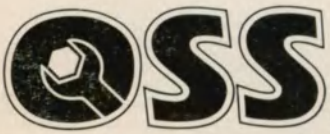
DOS XL™

Now Includes BUG/65



Precision
Software Tools

A USER'S GUIDE
and
REFERENCE MANUAL FOR

DOS XL version 2.30

A DISK OPERATING SYSTEM

designed for use with

ATARI HOME COMPUTERS

and compatible disk drives capable
of single and/or double density operation

This manual revised December, 1983

TRADEMARKS

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

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

Atari, Atari Home Computers, Atari Writer, Atari 810 Disk Drive and Atari 850 Interface Module are trademarks of Atari, Inc., Sunnyvale, CA.

PREFACE

DOS XL is the result of the efforts of several persons, and we believe that proper credit should be given. The original 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 2.0S) were written by Paul Laughton. 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.

SERVICE AND SUPPORT POLICIES

OSS has worked to bring you products which will give you years of service and enjoyment. As with any software or hardware product, though, errors or omissions can and do occur. You may rest assured that, if you have a problem, every reasonable effort shall be made to help you.

Generally, you may direct questions and problem reports about DOS XL to OSS. However, since DOS XL is only distributed as a licensed product, you must sign and return the OSS License Agreement included in your DOS XL package before we can respond to your inquiries.

If you have a quick question or simply a procedural problem, you may call the technical support staff at OSS. In order to allow our personnel time to answer letters, research your problems, and eat lunch, we must ask that you limit your technical calls to the hours of 9:30 AM to Noon or 1:30 PM to 4:30 PM (all times Pacific Time, please). Our technical support number is
(408) 446-3117.
Even though we have several phone lines, it is not unusual to find them all busy. Please be patient and try again.

Please understand that our support staff has only limited resources and may not be able to answer all your questions in a 5-minute phone call. So, if your problem is such that it is not easy to describe, you are invited to mail us a letter and include a diskette or computer printout detailing or demonstrating your difficulty. Please use the address given on the previous page.

Finally, if you feel the problem lies in the disk drive itself, you must call or write the manufacturer of your drive. Refer to the literature you received with your drive for the appropriate address and phone number.

ABOUT THIS EDITION

DOS XL is the latest in a series of Disk Operating Systems produced by Optimized Systems Software, Inc.

DOS XL version 2.3 is a direct successor to and completely file compatible with

Atari DOS 2.ØS
OS/A+ version 2.Ø
OS/A+ version 2.1
DOS XL version 2.2

This edition of the DOS XL manual has been issued as what we hope is both a user-friendly "guide" to the more commonly used features of the operating system as well as a true "reference manual" for the entire DOS XL Disk Operating System.

What parts of the DOS XL reference manual you read first should depend on your experience level and your purposes:

--If you will never program in any language, you may not need to read any more than chapters 1 through 3.

--If you are an Atari BASIC programmer, you should definitely read chapter 4 as well as the BASIC reference manual BEFORE you start using DOS XL with Atari BASIC.

--If you are an assembly language programmer, we would suggest reading the entire OS/A+ manual, paying special attention to chapter 1Ø.

--Finally, if you would like to automate DOS XL, allowing it to do several tasks for you while your computer is unattended, you need to read chapter 8.

Of course, regardless of your experience level or purposes, if you get tired of the restrictions of the DOS XL menu, you should read chapters 5 and 6. You might find them enlightening.

Whatever you choose to do, we hope that this guide and the reference manual will help you. Written suggestions about these manuals are always helpful and carry much more impact than verbal comments. Your letters are always welcome.

TABLE OF CONTENTS

Section 1: INTRODUCTION

This manual is an addendum to the manual for OS/A+, which DOS XL supercedes. At times throughout this addendum, you will be referred to specific sections of the OS/A+ manual for more information.

1.1 System requirements

DOS XL requires 32K of memory, and will work on ALL Atari computers. Although DOS XL is designed for all users, it enhances the usage of OSS SuperCartridges and/or Atari XL-series computers by extending program memory.

1.2 What is a DOS?

The purpose of DOS XL is to provide a way for your Atari computer to communicate with your disk drives, printer, and other peripherals. DOS XL 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 XL, MAC/65, BUG/65, Atari BASIC, etc.).
4. Pass control of the computer between the Operating System (DOS XL), Cartridges, and programs stored on disk.

DOS XL is the only DOS for Atari computers which lets you choose either an easy-to-use menu or a versatile command processor. When you use DOS XL in conjunction with the OSS SuperCartridge or the Atari XL computers, you also gain up to 5K of user memory.

1.3 Disk Files

Much like a record or a cassette tape can hold a number of songs, a single diskette can hold many distinct files of information (up to 64 files per diskette). These files can hold programs or data in text or other form. Unlike files stored on a cassette, each disk file must have a name associated with it.

The rules for valid DOS XL disk file names are:

- One to eight characters in length
- Optionally followed by a period and a one to three character extender.
- Only characters A-Z and Ø-9 are allowed.
- The first character must be a letter from A-Z.

Valid file names:	Illegal file names:	
GEORGE	NAMETOOLONG	
TEMP.ABC	TEMP.LONGEXTENSION	
PROG1.SAV	1PROG.SAV	(starts with digit)
SORT123	sort123	(lower case letters)
COPY.COM	BAD-CHAR	

The portion of the file name preceding the period is called the primary file name, and the optional portion of the file name following the period is called the extender. Although any combination of valid characters may be used for both sections of the file name, it is recommended that the extender be dedicated to indentifying the type of information contained in the file. The following extenders are suggested:

Extender:	Suggested usage:	Example:
SAV	"SAVE"d BASIC file	GEORGE.SAV
LIS	"LIST"ed BASIC file	PROG1.LIS
ACT	ACTION! source file	MIKE.ACT
M65	"SAVE"d MAC/65 file	SORT1.M65
OBJ	binary object file	BILL.OBJ
COM	DOS XL utility program	COPY.COM
EXC	DOS XL execute file	STARTUP.EXC
SYS	system program - reserved for DOS XL	

In most cases, file names must be preceded by a device specifier which tells the system on which drive to search for a particular file. The format of a device specifier is:

Dn:

(or)

D:

where n is a digit from 1 to 4, depending on how many drives you have. If you just specify D:, drive 1 is assumed (this is very useful if you only have one drive). Here are several examples of complete file names as you would type them into the computer:

File name:	Meaning:
D1:GEORGE	file GEORGE on drive 1
D2:MIKE.ACT	file MIKE.ACT on drive 2
D:TEMP.LIS	file TEMP.LIS on drive 1

In some cases file names 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, AB*.C?? will match ABX.CXX AB.CUR ABCDEF.CNN etc. Wild-card characters may be used in the following DOS XL menu commands: Files on Disk Copy Files Erase Files Protect Files Unprotect Files See section 3 for more information on these menu commands. See section 5 for more on "wild-cards".

1.4 Other devices

The Atari Personal Computer considers everything except the guts of the computer (i.e. the RAM, ROM, and processing chips) to be external devices. Actually, some of these "external devices" come with the computer (for example, the Keyboard and the Screen Editor). Some of the other devices are Disk Drive, Program Recorder (cassette), and Printer. When prompted for a file name by DOS XL, you need not always enter the name of a disk file. Other devices are referred to by names consisting 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 (e.g., D1: or D2:). The device name must be followed by a colon. The following is a list of device names which may be used under standard DOS XL:

C: The Program Recorder -- handles both Input and Output. You can use the recorder as either an input or output device, but never as both simultaneously.

D1: - D8: Disk Drive(s) -- handles both Input and Output. Unlike C:, disk drives can be used for input and output simultaneously. You are also required to specify a file name with this device, as previously mentioned.

NOTE: if you use D: without a drive number, D1: is assumed.

E: Screen Editor -- handles both Input and Output. The screen editor simulates a text editor/word processor using the keyboard as

input and the display (TV or Monitor) as output. This is the editor you use when typing in a BASIC XL program. When you specify no channel while doing I/O, E: is used because the channel defaults to Ø, which is the channal BASIC XL opens for E:.

K: Keyboard -- handles Input only. This allows you access to the keyboard without using E:.

P: Parallel Port on the 85Ø Module -- handles Output only. Usually P: is used for a parallel printer, so it has come to mean 'Printer' as well as 'Parallel Port'.

R1: - R4: The four RS-232 Serial Ports on the Atari 85Ø Interface -- handle both Input and Output. These devices enable the Atari system to interface to RS-232 compatible serial devices like terminals, plotters, and modems.

NOTE: if you use R: without a device number, R1: is assumed.

S: The Screen Display (either TV or Monitor) -- handles both Input and Output. This device allows you to do I/O of either characters or graphics points with the screen display. The cursor is used to address a screen position.

1.5 DOS XL command modes

A primary feature of DOS XL is its two modes of command entry. You, the user, may choose either a menu mode or a command processor. For those of you unfamiliar with other menu driven systems such as Atari DOS, a menu is simply a list of commands which appear on the screen. You need simply choose one of the options listed before you. If additional information is required, you are further prompted by the system for input. In this way, you need not remember the names of DOS functions; instead, you may simply select a command from the list.

The other input mode for DOS commands is the Command Processor mode, or "CP". In this mode, you are NOT shown a list of commands to choose from. Instead, you must invoke the DOS commands by name. Although this might at first be cumbersome, once several commands have been committed to memory, the command mode is much faster and easier to use. Also, certain advanced features of the DOS are available only from the "CP".

When DOS XL is shipped to you, it is set up so that you will be presented with the menu mode of command entry. It is recommended that this mode be used exclusively until you have gained some familiarity of the system. At that time you may, if you wish, read carefully sections 4 and 5 which discuss using the DOS XL command processor. After trying out the command mode, you may refer to section 8 in order to modify the system so that the command processor appears by default instead of the menu.

The following section summarizes the differences between the menu and command modes of DOS XL operation.

The DOS XL menu has the following advantages and disadvantages:

Advantages:
1) You do not need to remember the names of DOS commands. Instead, you are prompted at each step for the proper information.
2) Those of you already familiar with Atari DOS 2.Øs may find the DOS XL menu more comfortable and easier to use.

Disadvantages:
1) Using the menu with DOS XL uses about 2K more memory space. However, this is offset by the fact that DOS XL saves you 5K with a supercartridge.

The DOS XL CP (command processor) has the following advantages and disadvantages:

Advantages:
1) Once a few commands have been learned, the command mode is faster to use than the menu mode.
2) Those of you familiar with other operating systems such as Apple DOS, CP/M, UNIX, or OS/A+ will find the command mode more conventional and familiar than the menu mode.
3) Unlike the menu mode, the DOS XL command processor uses no extra user memory.

Disadvantages:
1) You are not prompted for input as in the menu mode. Therefore, you must learn several fundamental commands in order to utilize the power of the command mode.

1.6 Overview of DOS XL Architecture

DOS XL (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 output 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 below illustrates this process.

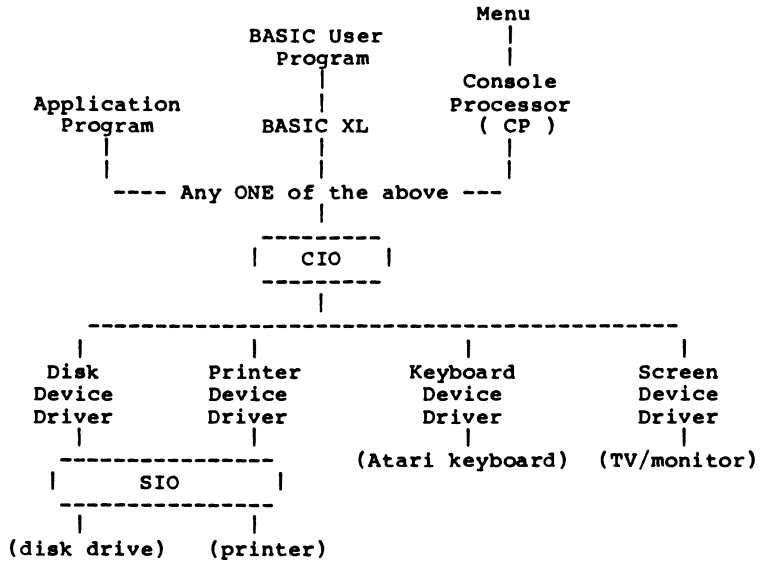


Figure 1-1
Overview of DOS XL

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, DOS XL itself (or for that matter any other DOS) is "added" to the default structure only if a disk drive is present at power-on time. Some 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 thought-out 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 (disk version), 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 810 Disk Drive to allow their usage on otherwise improved products.

As you might have noted in Figure 1-1, the menu and CP (Console Processor) are NOT priveleged parts 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, and the menu provides an even simpler access into CP and from there to the rest of the system.

1.7 Glossary of terms

The following terms are used throughout the rest of this manual. Their definitions are included here so that you may familiarize yourself with them before proceeding. Please refer back to this section to clarify concepts introduced later in the manual.

Term:	Definition:
DOS	An acronym for Disk Operating System.
boot booting	The process of loading DOS XL or another system into memory when you power on your computer.
master diskette master disk	The DOS XL diskette you received with your purchase of DOS XL, or a duplicate thereof.
bootable diskette	Any diskette which contains the file DOS.SYS. Such a disk can be used to boot DOS XL into memory. To take advantage of the you also need DOSXL.SYS on that diskette.
file name	The string of characters used to refer to a specific file on a specific drive (e.g., D:GEORGE).
filespec	A file name which may contain the wild-card characters '?' and '*' (e.g., D:**.*, D2:*.CO?).
prompt	Any message from DOS XL instructing or asking you to type a response.
[RETURN]	The key located on the right side of the keyboard of your Atari computer, marked "RETURN".
utility utility program	A program which is used by DOS XL to perform a disk operation. Examples include COPY.COM, which copies files; INIT.COM, which initializes disks; etc. From the viewpoint of DOS XL, even languages such as BASIC XL and Atari BASIC are utilities.

Section 2: GETTING STARTED WITH DOS XL

2.1 Booting your DOS XL master disk

The first operation you should perform after opening your DOS XL package is to fill out your license agreement and mail it to OSS. This action puts you on our mailing list for a quarterly (usually) newsletter announcing new products, updates to existing products, solutions to problems with our products, and answers to common user questions.

Once this is done, you should determine whether you wish to use the single density or double density version of DOS XL. In truth, you will probably find occasion to use both versions, so OSS has provided you with a two-sided master diskette. One side of your DOS XL master diskette contains a single density version of DOS XL; the other side contains a double density version of the same thing. But why use one version over the other? There may be many reasons for choosing one version over another, but allow us to give you what we feel are a few good reasons:

SINGLE DENSITY: To be compatible with diskettes created or designed for Atari 810 disk drives, you should use the single density version. Single density DOS XL is completely file compatible with Atari DOS 2.0s. All operations which work with Atari DOS will generally work with single density DOS XL.

Are there exceptions? Yes. Several companies produce self-booting disks (that is, you simply put them in the disk drive and turn on the computer) which make calls or references to routines or addresses internal to Atari DOS. There is nothing we can do to make DOS XL compatible with these disks! On the other hand, this is not really a problem, since (as we mentioned) these disks are generally self-booting (implying that they include a copy of Atari DOS on their disk). To use these diskettes, do nothing special. Simply follow their manufacturers' directions.

DOUBLE DENSITY: Generally, most programs written with cartridge-based languages will work fine with double density DOS XL. This usually includes programs written in Atari BASIC, PILOT, LOGO, OSS BASIC XL, MAC/65, C/65, and more. Of course, if you yourself have written the program and have not made assumptions about the size and type of disk, your programs will run correctly.

What doesn't work in double density? Programs which assume that sectors always contain 128 (or 125) bytes. Programs which are self-booting and which have special "protection" schemes. Programs which bypass DOS entirely. As mentioned above, though, these programs are usually distributed on self-booting disks, so you need do nothing special with them.

In any case, once you have decided which density disk to boot, perform the following steps:

1) Connect your disk drive and any other peripherals to your Atari computer following the manufacturer's instructions.
2) Turn on your peripherals and your monitor or television.
3) Insert the DOS XL master disk you have chosen to use (i.e., either single or double density) into the disk drive (drive 1 if you have more than one drive).
4) Now plug a cartridge, if you wish, into the cartridge slot.
5) Finally, turn on your Atari computer.

The disk drive will be accessed, and after a time the DOS XL copyright message will appear at the top of the screen. Then, a message will begin to appear, line by line, which begins with "Welcome to DOS XL...". The last line from the startup file is just "MENU". This is a command which instructs DOS XL to load the DOS XL menu. After a few more seconds, the menu program will finish loading, the screen will again clear, and the DOS XL menu will appear.

| The scrolling text which appears on the screen comes from a special file on the disk which is named "STARTUP.EXC". In section 8.3, you can learn how to modify these messages or omit them entirely.

2.2 Selecting menu options

You should now be presented with the DOS XL menu on the screen. It should look like this:

```
        DOS XL MENU   version 2.30
          copyright (c) 1983  OSS, Inc.

        Files on Disk     Protect Files
        To Cartridge      Unprotect Files
        Copy Files        Rename File
        Duplicate Disk    Save Binary
        Erase Files       Load Binary
        Initialize Disk   Go to Address
        Xtended Command   Quit to DOS XL

        Enter your selection.
```

NOTE: If the version number shown on your menu is 2.31, 2.32, etc., there is no cause for alarm. OSS reserves the last digit of the revision number as a "patch" number. Patches never include major documentation or feature changes but instead only fix bugs found in prior releases.

Although this diagram does not show it, the first characters of each of the 12 commands in the list are in inverse video. These characters are those which you would type to select a command. For example, to invoke the "Rename File" command, you would simply type 'R', to the above prompt. For most of the menu commands, extra input is required. The menu will prompt you with appropriate messages whenever you are required to input new data.

Whenever you are prompted by the DOS XL menu to enter a filename or a filespec, you don't always have to specify the device name for the drive. If you do not specify a device name (i.e., D:, D2:, etc.), drive 1 is assumed (D1:). For example, if you typed "GEORGE" as a filename, the menu would assume that you meant "D1:GEORGE".

CAUTION: In general, you may not omit the drive specifier on names entered while using cartridge based products such as Atari BASIC or OSS BASIC XL. Except when working with the menu or DOS XL's Console Processor, you usually need to specify the entire file name (and often must enclose it in quotes, as in Atari BASIC).

2.3 Files on the DOS XL master disk

In order to insure the integrity of your DOS XL master diskette, you should view the names of the files contained on it. The DOS XL menu of commands should be visible on the screen, and the menu will prompt you:

Enter your selection.

The first command in the menu is "Files on disk". This command allows you to view the names of disk files. First insure that your DOS XL master diskette is still in drive 1. Then select this command as follows:

You type: F

When the menu responds by prompting with "Filespec:", simply type [RETURN].

The list of files on your master disk will appear and should look like this (the numbers may differ a bit):

```
* DOS      SYS 046
* DOSXL    SUP 046
* DOSXL    XL  060
* MENU     COM 025
* CLRDSK   COM 023
* COPY     COM 075
* DO       COM 003
* DUPDBL   COM 011
* DUPDSK   COM 011
* INIT     COM 006
* INITDBL  COM 023
* NOVERIFYCOM 001
* RS232    COM 001
* RS232FIXCOM 002
* SDCOPY   COM 086
* VERIFY   COM 001
* SYSEQU   ASM 022
* MEM      LIS 066
  STARTUP  EXC 003
160 FREE SECTORS
```

If the list of files looks essentially the same, but the numbers on the right side are about half what is shown, you've booted the Double Density version of DOS XL! In particular, if the first line reads `* DOS SYS 023` you can be sure that you are working in double density. The number on the right indicates the file size in sectors. Since double density sectors hold a little more than twice as much as single density sectors, it makes sense that the file sizes are smaller. Also, notice how many more "FREE SECTORS" you have in double density. THIS is why you bought a double density drive!

Each of the files has a primary name and an extension. For example, the file COPY.COM has primary name COPY and extension COM. Note that in the file listing the period is not shown. Instead, there may be one or more spaces between the primary name and the extension. This format appears ONLY in the file listing. You may never use this form for specifying files. If you are using file extensions, you must use a period with no intervening spaces, as in "D:TEST.LIS".

Notice that all of the files on the master disk except the file "STARTUP.EXC" are preceded by an asterisk in the directory listing. An asterisk preceding a file implies that it has been protected from modification or erasure through the use of the DOS XL menu command "Protect Files". This method can also be used to protect your own files from change or deletion through accidental use of one of the DOS XL commands (see section 3.9 for further information).

2.4 Backing up your DOS XL master disk

Now that you have successfully booted your master disk, you should make a backup copy. If your DOS XL master disk should ever fail to operate properly, you may then transfer the files on your backup copy to the master disk, thus restoring the master to a working state.

First, for safety's sake, place a write protect tab over the notch on your DOS XL master disk, if it does not already have one. This protects the entire diskette from being written to in any way. You should also use write protect tabs to protect your program disks from accidental change when you do not need to write to them.

At this point, the DOS XL menu of 12 commands will be displayed on the screen. The following prompt will appear with the cursor below it:

```
Enter your selection.
```

You type: D

This is the command for Duplicate Disk. You will then be asked:

```
Double density?
```

If the size of DOS.SYS indicated in the file directory was Ø46 (sectors), then you booted a single density diskette, so

You type: N [RETURN]

However, if the size of DOS.SYS indicated in the file directory was Ø23 (sectors), then you booted a double density diskette, so

You type: Y [RETURN]

Either the DUPDSK.COM (short for DUPlicate DiSK) or DUPDBL.COM (short for DUPlicate DouBLe density disk) utility program will be read into memory, depending on whether you answered N or Y to the last prompt. The next prompt is:

```
Source disk (1,2,3,4) :
```

Normally, you will copy from drive 1 so,
You type: 1 [RETURN]

To the prompt:

```
Destination disk (1,2,3,4) :
```

You type: 1 [RETURN]

SPECIAL NOTE: If you have two disk drives, please refer to section 7 of this manual for information on how to configure them for various purposes. Once you are sure they are configured properly, you can then use drive 2 as your destination disk and the duplication process will proceed much more speedily.

You will be asked:

```
Format destination disk (Y/N)?
```

Most blank diskettes are unformatted. That is, they are not yet prepared to hold disk files. In order to copy files or diskettes to blank diskettes, they must first be formatted. Therefore,
You type: Y [RETURN]

At this point, the DUPDSK or DUPDBL utility will ask:

```
Insert source disk into drive 1
And hit RETURN when ready
```

Your master disk should still be in drive 1 at this point, so just type:

[RETURN]

The light on the front of your disk drive will come on, and the DUPDSK utility will respond by saying:

```
Reading source disk
```

After a while, you will be prompted:

```
Insert destination disk into drive 1
And hit RETURN when ready
```

The DUPDSK utility has read as much as possible of the source disk into memory. At this time, remove your DOS XL master disk from drive 1 and insert a blank diskette. When this has been done,
You type: [RETURN]

The program will respond:
Formatting destination disk
and, after a while,
Writing destination disk

Most of the time, the total information on a diskette is too large to hold in your Atari's memory at one time. This is the case for your DOS XL disk. Therefore, you will be prompted to repeatedly insert your source and destination disks (the DOS XL master and the blank diskette, respectively) until the duplication is complete. Follow these prompts carefully until the DUPDSK utility responds:

Copy same disk again (Y/N)?
You type: N [RETURN]

The DOS XL menu will then prompt:

Hit RETURN for menu
You type: [RETURN]

At this point, you have successfully made a copy of your DOS XL master disk. First place a write protect tab over the notch on the copy so that it is never accidently written to. Label the disk appropriately and store it in a safe place so that you can use it if ever your master disk fails to work properly.

2.5 Entering the cartridge

At this point, if you wish to enter the cartridge to use BASIC, BASIC XL, ACTION!, or other cartridge based products, you must use a DOS XL menu command.

If you have not already plugged in the desired cartridge, you should turn off your Atari computer and insert it. Then insure that the DOS XL master disk is in drive 1 and turn on the power. The boot process will again take place, putting you in the menu.

To enter the cartridge you must use the "To cartridge" command in the DOS XL menu. To use that command,
You type: T

You will then be in the cartridge, ready for programming.

If you are using the Atari BASIC cartridge, please refer to section 4 of this manual for details on the commands which may be used from BASIC to access the disk. For those of you already familiar with Atari BASIC under Atari DOS 2.Øs, please note that disk access is identical under DOS XL.

Section 3: THE DOS XL MENU

The DOS XL menu was designed to be easy to use while allowing you access to the full power of the Atari computer.

For those of you who have previously used Atari DOS, here is a summary of the differences between the DOS XL menu and the Atari DOS menu:

1) Loading DOS - The Atari DOS menu must be loaded in from the disk whenever you return to DOS from a cartridge. Since we felt that this process was too slow and cumbersome, we made sure that the DOS XL menu MAY be kept "resident" (in memory) at all times. This does occupy about 2,000 bytes more memory, but this is more than offset if you are using DOS XL with a SuperCartridge. If you wish to utilize the memory taken up by the menu, you may use the DOS XL command processor, which does not use that memory.

2) MEM.SAV - Atari DOS supports the use of a special file in which to save user memory while accessing DOS functions. DOS XL neither supports nor needs a MEM.SAV file.

3) Single key commands - The DOS XL menu needs only a single key to access commands, whereas Atari DOS requires a carriage return after the command letter.

3.1 Entering Commands

Whenever the menu is entered, the following list of commands will appear on the screen:

```
DOS XL MENU   version 2.30
  copyright (c) 1983  OSS, Inc.

Files on Disk     Protect Files
To Cartridge      Unprotect Files
Copy Files        Rename File
Duplicate Disk    Save Binary
Erase Files       Load Binary
Initialize Disk   Go to Address
Xtended Command   Quit to DOS XL
```

When the DOS XL menu is visible on the screen, you are prompted:

```
Enter your selection.
```

At that point, you should type the first letter of any of the DOS XL menu commands listed on the screen. If more input is required to complete the command, the menu will prompt you for more information. Unless the command loads a utility program, you may abort a command at any time by hitting the ESCAPE key on the upper left of your keyboard. If the command loads a utility program (the commands "Copy Files", "Duplicate Disk", and "Initialize Disk"), you may have to press SYSTEM RESET in order to abort the command.

The following sections describe each menu command in detail. The commands are presented in alphabetical order, NOT the order in which they appear in the menu.

3.2 C - Copy Files

CP equivalent: COPY

The copy command allows you to transfer files between diskettes or to different files on the same diskette. The "Copy Files" command is most useful for copying one or a few files from one diskette to another. If you desire to transfer all or most of the files on a single diskette to another diskette, you should use the "Duplicate Disk" command instead as it will perform the operation much more rapidly.

To use the copy command, select 'C' when the menu prompts you for a command selection. At that time, DOS XL will check for the COPY.COM file on the diskette in drive 1. This is the utility program which performs file transfers. If DOS XL does not find the COPY.COM program, you will be prompted to insert your DOS XL master disk as follows:

```
Copy Files
Insert MASTER disk and hit RETURN
```

If you receive this prompt, take out any disk in drive 1 and insert your DOS XL master disk and press the RETURN key.

The menu will then prompt:

```
Copy Files
From file:
```

At this point, you should respond with a filespec specifying the file or files to be copied (e.g., D:GEORGE, D2:JUNK.LIS, etc.). For example, if you want to transfer the contents of the file "PROG1" on drive 1 to another diskette, you should type "D1:PROG1".

Notice that wild-cards may be used to refer to files using the COPY utility (e.g., TEMP.*, AB??.COM, etc.). As a special case, if you wish to copy all files on a disk to another disk, just use a filespec of Dn:, where n is the source drive number (e.g., D1:).

The COPY utility will then prompt:

```
To file:
```

You should respond with the destination filespec. In most cases you will want to transfer files from one diskette to another without changing their names. In

this case you may refer to the destination filespec as just Dn:, where n is the destination drive number (e.g., D:, D1:, D2:). In the above example, if you wanted to copy "PROG1" to a different diskette and you own only 1 drive, you should type "D1:".

You will then be asked:
        Single Drive?

If you own only a single drive as in the above example, or if you are performing this copy to another diskette in the same drive,
You type: Y [RETURN]

In any other case,
You type: N [RETURN]

The COPY.COM utility program will then be loaded from the diskette, and you will be prompted:
        Insert disk(s) to be copied
        and hit RETURN when ready

Remove your master disk and insert your source disk. If you own more than one drive and are copying to a second drive, insert your destination disk into the proper drive.
You type: [RETURN]

Before each file is copied, you will be asked:
        Copy
          Dn:filename
        to Dn:filename?

If you wish to copy that particular file,
You type: Y [RETURN]

Otherwise,
You type: N [RETURN]
If you choose not to copy a file, a message will be printed to the screen verifying that the file was not copied.

At this point the source file will be read into memory. If you are copying to another disk on the same drive, you will then be prompted to insert the destination disk as follows:
        Insert 'to' disk and hit RETURN

If the destination file already exists, you will be asked:
        'To' file already exists
        OK to overwrite?

If you wish to replace the old file with the source file,
You type: Y [RETURN]

Otherwise,
You type: N [RETURN]

| If the destination file has previously been guarded against modification by using the DOS XL menu "Protect Files" command (i.e., the file is preceded by an asterisk in the "Files on Disk" listing of the disk), the COPY utility program will not be able to overwrite that file. The protection must first be removed using the "Unprotect Files" command before that file may be written to.

The COPY utility reads as much as possible of the source file into memory at one time. If the source file is too large to fit into memory and you are copying on a single drive, you will again be prompted:

Insert 'from' disk and hit RETURN

Re-insert your source disk and continue to carefully follow the directions of the prompts until the entire file is copied.

When a file has been completely copied, a verification message will be printed on the screen. When all files have been copied, you will be prompted:

Hit RETURN for menu

To return to the list of menu commands,
You type: [RETURN]

| The "Copy Files" command should not be used to copy from single to double density diskettes if you own only one disk drive. Instead see section 7 for operations involving multiple densities. In particular, section 7.3 discussing single to double density copies on a one drive system. See also section 6.16 and be sure to read all of section 7.

3.3 D - Duplicate Disk

CP equivalents: DUPDSK
DUPDBL

This command allows you to copy quickly the entire contents of a diskette to another diskette. If you wish to copy only one or just a few files from one diskette to another, or if you need to preserve some of the files already on the disk you wish to copy TO, the "Copy Files" command should be used instead.

The "Duplicate Disk" command writes entirely new information to the destination diskette, thus erasing completely all files which previously existed there. Carefully select the desired destination diskette to avoid accidently destroying your program disks.

To select the Duplicate Disk command, type 'D' when you are prompted to enter a command selection.

You will then be asked:

```
         Duplicate Disk
        Double density?
```

If your source disk was formatted under single density,
You type: N [RETURN]

If the source disk is double density,
You type: Y [RETURN]

The duplicate disk utility program will be read into memory. The next prompt is:

```
        Source disk (1,2,3,4) :
```

Normally, you will copy from drive 1 so,
You type: 1 [RETURN]

You will be prompted:

```
        Destination disk (1,2,3,4) :
```

If you have only a single drive, or you wish to use drive 1 as your destination drive,
You type: 1 [RETURN]

If you wish to use a drive other than 1 for a destination drive,
You type: n [RETURN]

where n is the number of the desired destination drive.

You will be asked:
        Format destination disk (Y/N)?

Most blank diskettes are unformatted. That is, they are not yet prepared to hold disk files. In order to copy files or diskettes to blank diskettes, they must first be formatted. Therefore,
You type: Y [RETURN]

At this point, the DUPDSK utility will ask you to:
        Insert source disk into drive 1
        And hit RETURN when ready

If you specified a destination drive different from the source drive, you will be prompted:
        Insert source disk into drive 1
        Insert destination disk into drive n
        And hit RETURN when ready

Insert the proper source (and destination disk, if using 2 drives) into the proper drive, and
You type: [RETURN]

The light on the front of your disk drive will come on, and the DUPDSK utility will respond by saying:
        Reading source disk

If the destination drive is the same as the source drive, you will be prompted:
        Insert destination disk into drive n
        And hit RETURN when ready

At this time, remove your source diskette from drive 1 and insert a blank diskette. When this has been done,
You type: [RETURN]

The program will respond:
        Formatting destination disk
and, after a while,
        Writing destination disk

Most of the time, the total information on a diskette is too large to hold in your Atari's memory at one time. This is the case for your DOS XL disk. Therefore, you will be prompted to repeatedly insert your source and destination disks until the duplication is complete. Follow these prompts carefully until the DUPDSK utility responds:
        Copy same disk again (Y/N)?

You type: N [RETURN]

At this point, you will be prompted:
Hit RETURN for menu

To return to the DOS XL menu,
You type: [RETURN]

3.4 E - Erase Files

CP equivalent: ERAse

The "Erase Files" command allows you to delete one or more files from a diskette. This command should be used with care, for erased files cannot easily be recovered, if at all.

If you use the "Erase Files" command to attempt to erase a file which has previously been protected (i.e., the file name is preceded by an asterisk in the directory listing), you will be given the error message "FILE PROTECTED". If you desire to erase this file, you must first remove the protection by using the "Unprotect Files" command. Note that protecting files is an excellent way of guarding against accidental erasure.

To use this command, select 'E' when the menu prompts you for a command selection. The menu will then prompt:

```
Erase Files
Filespec to erase:
```

You should respond with the name of the file you wish to erase. If you wish to erase a group of files, you may use wild-card characters in the filespec. However, be very sure you know what you are erasing.

You will then be asked:

```
Are you sure?
```

If you feel the filespec you entered was correct,
You type: Y [RETURN]

If you wish to abort the "Erase Files" command,
You type: N [RETURN]

If you answered 'Y', all files matching the selected filespec will be removed from the diskette. The menu will then prompt:

```
Hit RETURN for menu
```

To return to the menu of commands,
You type: [RETURN]

3.5 F - Files on Disk

CP equivalent: DIRectory

The "Files on Disk" command allows you to view the names of any or all files on a diskette.

To use this command simply select 'F' when prompted by the menu for a command selection. Then insert the desired diskette into one of your disk drives (or drive 1 if you have only one drive). At that point, the menu will prompt:

```
 Files on disk
Filespec:
```

The filespec required instructs DOS XL which files on the disk to look for and display. The following table gives some examples of filespecs and the corresponding lists of files they display:

Filespec:	Files listed:
GEORGE	The file having the name GEORGE, if such a file exists.
JUNK.SAV	The file having the primary name JUNK and the extender SAV, if such a file exists.
AB?	Any file not having an extender whose name is three characters long where the first two are AB. This filespec matches ABC, ABX, AB1, etc.
CAT*	Any file whose name begins with CAT. The filespec matches CAT, CATCHER, CATTLE, etc.
JOHN.??X	Any file whose primary name is JOHN and whose extender is three characters long ending in X. This matches JOHN.ABX, JOHN.XXX, etc.
**.*	All files on the diskette. This filespec may be abbreviated by just [RETURN].

D1:	All files on the diskette in drive 1.
D2:	All files on the diskette in drive 2.

3.6 G - Go to Address

CP equivalent: RUN

This command allows you to pass control of your computer to a machine language program already residing in your Atari computer's memory. This program should have previously been loaded into memory using the DOS XL menu "Load File" command, or an equivalent method.

To use the "Go to Address" command, type 'G' when the menu promt "Enter you selection." appears. At that time, the menu will prompt:

```
Go to Address
Address:
```

You should respond with the hexadecimal address of the location in memory you desire to jump to. For example, if a machine language program resides at location $5000 (the dollar sign indicates hexadecimal), you would respond with "5000". Note that although the number is a hexadecimal value, you should not precede it with a dollar sign when you enter it.

Be sure that the address you enter is correct; for, in general, if you pass control to a location in memory which does not contain the desired machine language program, control of your computer will be lost and the keyboard will "hang". In some cases, hitting the SYSTEM RESET key on your computer's keyboard will return control to you. Most of the time, however, you will be forced to turn off the power to your computer and repeat the boot process.

At this point, control will be passed to the machine language routine located at the desired address. If that routine returns to the menu with a 6502 RTS instruction, you will be asked:

```
Hit RETURN for menu
```

To return to the menu of commands,
You type: [RETURN]

3.7 I - Initialize Disk

CP equivalent: INIT

The "Initialize Disk" command allows you to format blank diskettes so that you may use them to store program and data files. If you wish to create a bootable diskette rather than just a data diskette, you will normally want to duplicate your DOS XL master disk. In this case you should use the "Duplicate Disk" command rather than the "Initialize Disk" command. If you want to duplicate any of your diskettes using the "Duplicate Disk" command, you do not need to format them first using the "Initialize Disk" command, for the "Duplicate Disk" utility will perform the format operation if you desire.

The "Initialize Disk" command writes entirely new information to the desired diskette, thus erasing completely all files which previously existed there. Carefully select the diskette to initialize to avoid accidently destroying your program disks.

To use this command, select 'I' when prompted for a command selection. At that point, DOS XL will check for the presence of the INIT.COM utility on the diskette in drive 1. If it is not there, you will be prompted:

```
        Initialize Disk
    Insert MASTER disk and hit RETURN
```

after which you should insert your DOS XL master disk and hit the RETURN key.

Then the INIT utility program will be loaded into memory and you will be present with the 4 options of the INIT program. They are:

1. Format disk only
2. Format disk and write DOS.SYS
3. Write DOS.SYS only
4. Exit to DOS XL

CAUTION: The I (INIT) option may normally be used ONLY to initialize diskettes of the same density as the master diskette you have booted. See section 7 of this manual for information on other options.

| Normally, you should use the "Duplicate Disk" command to copy your DOS XL master to create a spare bootable disk. However, option 2 (and option 3 when used with an already formatted diskette) of the "Initialize Disk" command may be used to create a bootable disk.
|
| Do NOT, however, use either option 2 or 3 if you have booted the system with the file "DOSXL.SYS" present and active (that is, if you have followed the instructions of section 8.1 for using an extended memory DOS system).
|
| If you are using an extended memory DOS configuration, we suggest that you boot your original master diskette (or a direct duplicate thereof) before using option 2 or 3 of this command.
|
| Comment: After using option 2 to create a bootable disk, the DOSXL.SYS file should be Copied onto that disk if it is to be used with an OSS SuperCartridge.

If you wish to create a bootable disk,
You type: 2 [RETURN]

If you wish to create just a data disk,
You type: 1 [RETURN]

You will be asked:
Drive (1,2,3,4):

You should respond with the desired drive number (always 1, if you have only one drive).

You will be asked:
Option n drive n - Are you sure (Y/N)?

If your are happy with your entries so far,
You type: Y [RETURN]

Otherwise,
You type: N [RETURN]

If you typed 'Y', the specified command will be executed. You will again be presented with the 4 options.

If you have more disks to initialize, repeat the above steps. Otherwise,
You type: 4 [RETURN]

You will be returned to the DOS XL menu.

3.8 L - Load Binary

CP equivalent: LOAd

The "Load Binary" command allows you to read a binary file from disk into the memory of your Atari computer. This command can be used to load binary object of assembly language programs, or binary data to be used by such programs. The file you wish to load should have previously been written to disk using the DOS XL menu "Save File" command, or an equivalent method.

Do NOT use this command to load Atari BASIC or BASIC XL programs into memory. Instead, just use the LOAD command from the BASIC cartridge (i.e., after you have been given the "READY" prompt). See section 4.6 of this manual for more information.

To use the "Load Binary" command, type 'L' when prompted to enter your command selection. the menu will then prompt:

```
        Load Binary
      Filename:
```

You should respond with the name of the previously saved file you wish to load. For example, if you wish to load into memory the file "FILE1.OBJ" on drive 1, you should type "D:FILE1.OBJ".

At this point, DOS XL will access the disk to read in the binary file. You will then be asked:

```
      Hit RETURN for menu
```

To return to the DOS XL menu of commands,
You type: [RETURN]

3.9 P - Protect Files

CP equivalent: PROtect

In many cases you will have created files on your disks which you know you will hardly ever need to modify. There is a way to guard these files so that you need not worry about accidently deleting or modifying their contents. The "Protect Files" command allows you to protect files from renaming, erasure, or modification. These files will then be preceded by an asterisk in a directory listing when you use the "Files on Disk" command. If in the future you desire to remove the protection afforded by this command, you should use the "Unprotect Files" command.

To use this command, select 'P' when the menu prompts you for a command selection. The menu will then prompt:

```
Protect Files
Filespec to protect:
```

You should respond with the name of the file you wish to protect. If you wish to protect a group of files, you may use wild-card characters in the filespec.

At this point the disk will be accessed and the files will be protected. The menu will then prompt:

```
Hit RETURN for menu
```

To return to the menu of commands,
You type: [RETURN]

3.10 Q - Quit to DOS XL

CP equivalent: none

The "Quit to DOS XL" command is used to pass control from the DOS XL menu to the DOS XL command processor. Although almost all the functions you need from DOS may be accomplished from the DOS XL menu, certain commands and features are accessible only from the command processor mode.

To use this command, type 'Q' when prompted by the menu to enter a command selection. At that point, control will be transfered to the command processor mode.

Whenever you enter the DOS XL command processor, the following message will appear on the screen:

```
DOS XL - Atari version 2.30
  copyright (c) 1983 OSS, Inc.

D1:
```

The "D1:" which appears to the left of the cursor is the prompt for the command mode. In this mode you are expected to type in a complete command line rather than simply a command selection. For example, to load the DUPDSK.COM utility (for duplicating single density diskettes), the DOS XL command line is "DUPDSK", rather than a single character as in the menu mode.

There are two major types of commands which you can use when the D1: prompt appears, Intrinsic Commands and Extrinsic Commands. As a user, the only real difference between these two types is that a master disk (or, sometimes, a subset thereof) must be in place in drive 1 in order to use an extrinsic command. Section 5 of this manual details the Intrinsic Commands. Section 6 describes the usual Extrinsic Commands.

It is also possible to write your own commands to be used from the DOS XL command processor or the "Xtended Command" menu function. For more information on this capability, refer to section 10 of this manual.

Certain features of DOS XL such as batch processing are available only from the DOS XL command processor. For information on batch processing and execute files, please refer to section 9 of this manual.

In order to return to the DOS XL menu from the command processor, insert your DOS XL master disk into your disk drive (or drive 1, if you own more than one drive). Then, from the D1: prompt,
You type: MENU [RETURN]

The DOS XL menu program will be loaded and executed.

3.11 R - Rename File

CP equivalent: RENname

The "Rename File" command may be used to change the file name associated with a file of information. This command does not alter or delete any information contained in the file. Rather, the file will only show up with a different name in the directory listing when using the "Files on Disk" command.

If you attempt to rename a file which has been protected against modification (i.e., the file name is preceded by an asterisk in the directory listing), you will be given the error message "FILE PROTECTED". If you desire to rename this file, you must first remove the protection by using the DOS XL menu command, "Unprotect Files".

To use the "Rename File" command, select 'R' when the menu prompt, "Enter your selection." appears. You will then be asked:

```
Rename File
Old name:
```

You should respond with the current name of the file whose name you wish to change. For example, if you want to change the name of the file "D:GEORGE" to "D:PROG1", you should type "D:GEORGE".

The menu will then respond,

```
New name:
```

At this point you should type the new name you wish the file to have. In the above example, you should type "PROG1" at this time. Notice that you must NOT use a device specifier (i.e., D:, D2:, etc.) in the new name; you should type just "PROG1", not "D:PROG1".

You will then be asked,

```
Are you sure?
```

If you are satisfied that you have entered both file names correctly,
You type: Y [RETURN]

If instead you wish to abort the rename operation,
You type: N [RETURN]

If you answered with 'Y', the designated file will be renamed, and you will be prompted:

```
        Hit RETURN for menu
```

To return to the list of menu commands,
You type: [RETURN]

3.12 S - Save Binary

CP equivalent: SAVe

The "Save Binary" command allows you to write a portion of your Atari computer's memory to a disk file. This command can be used to save to disk binary object of assembly language programs, or binary data to be used by such programs.

Do NOT use this command to save Atari BASIC or BASIC XL programs from memory. Instead, just use the SAVE command from the BASIC cartridge (i.e., after you have been given the "READY" prompt). See section 4.12 of this manual for more information.

If you attempt to save binary data to a file which has been protected against modification (i.e., the file name is preceded by an asterisk in the directory listing), you will be given the error message "FILE PROTECTED". If you desire to rename this file, you must first remove the protection by using the DOS XL menu command, "Unprotect Files".

To use the "Save Binary" command, type 'S' when prompted to enter your command selection. the menu will then prompt:

```
Save Binary
Filename:
```

You should respond with the name you wish the saved file to have. For example, if you wish to write memory from locations $4000 to $4100 (the dollar signs indicate hexadecimal addresses) to the file "FILE1.OBJ" on drive 1, you should type "D:FILE1.OBJ".

It is recommended that binary object file names have either the extension "OBJ", or "COM". In the former case, "OBJ" would indicate that the file was an assembly language OBJect file for a program or data. The second extension, "COM", indicates that the program is a system utility program which was either included with your DOS XL master disk or written by you or another user.

At this point you will be prompted:

```
Starting address:
```

You should respond with the hexadecimal value of the first address you wish to write to disk. In the above example, the starting address was $4000 so you should type "4000". Note that although the value is hexadecimal, you should not precede the number with a dollar sign.

The menu will then prompt:

Ending address:

You should respond with the hexadecimal value of the last address you wish to save. In the previous example, you should enter "4100".

At this point, DOS XL will access the disk to write out the binary file. You will then be asked:

Hit RETURN for menu

To return to the DOS XL menu of commands,
You type: [RETURN]

3.13 T - To Cartridge

CP equivalent: CARtridge

This command allows you to enter a cartridge, if one has been inserted.

If you are using the Atari BASIC cartridge, please refer to chapter 4 of this manual for information on commands which may be used from BASIC to access the disk. For those of you already familiar with Atari BASIC under Atari DOS 2.0s, please not that disk access is identical under DOS XL.

To use this command, select 'T' when prompted by the menu for a command selection. At that time, you will enter the cartridge and see the familiar READY prompt of BASIC, or the prompt for the particular cartridge you are using. If no cartridge was inserted, the error message NO CARTRIDGE will be displayed.

If the "To Cartridge" command is used after any of the following commands are selected: Copy Files Duplicate Diskette Initialize Diskette Xtended Command Load Binary a coldstart will be performed by the cartridge, thus erasing any program which was in memory. Therefore, if you wish to go to the menu to execute any of these commands, remember to first write any program you are working on to disk. This is accomplished in Atari BASIC or OSS BASIC XL by using the SAVE command in the BASIC cartridge.

3.14 U - Unprotect Files

CP equivalent: UNProtect

The "Unprotect Files" command allows files to be renamed, erased, or modified, thus removing protection applied by the "Protect Files" command. These files will no longer then be preceded by an asterisk in a directory listing when you use the "Files on Disk" command.

To use this command, select 'U' when the menu prompts you for a command selection. The menu will then prompt:

```
Unprotect Files
Filespec to unprotect:
```

You should respond with the name of the file you wish to unprotect. If you wish to unprotect a group of files, you may use wild-card characters in the filespec.

At this point the disk will be accessed and the files will be unprotected. The menu will then prompt:

```
Hit RETURN for menu
```

To return to the menu of commands,
You type: [RETURN]

3.15 X - Xtended Command

CP equivalent: none

This command may be used to pass a command line to the DOS XL command processor. Although almost all the functions you need from DOS may be accomplished from the DOS XL menu, certain commands and features are accessible only from the command processor mode. The "Xtended Command" function of the DOS XL menu may be used to access from the menu those commands available only from the command processor. Please refer to sections 5 and 6 for information about the commands and features of the DOS XL command processor.

To use the "Xtended Command" function, select 'X' when prompted by the menu "Enter your selection.". At that time, the menu will prompt:

```
Xtended Command
Command:
```

You should respond with the DOS XL command you wish to have executed. For example, if you wish to use the "RS232" command, you should type "RS232".

Many of the DOS XL commands accessible by the "Xtended Command" function perform their operations by loading utility programs on the DOS XL master disk. If you wish to use a command which employs a utility program (any "extrinsic" command, see this manual's section 6), you should insure that your DOS XL master disk is first inserted into drive 1.

At this time, the desired command will be passed to the DOS XL command processor and executed. When that is finished, you will be prompted:

```
Hit RETURN for menu
```

To return to the DOS XL menu of commands,
You type: [RETURN]

Section 4: Atari BASIC and DOS XL

When you boot DOS XL from your OSS Master Diskette, you are (after some preliminary messages) presented with a MENU of available options. In section 2.5 and, again, in section 3.13 you read about the T ("To cartridge") command. If you booted DOS XL with your BASIC XL or Atari BASIC cartridge in place (or if you have an Atari XL computer with BASIC built in), you may simply press the "T" key and you should receive BASIC's READY prompt. If you do not receive a READY prompt after pressing "T", perhaps you forgot to insert your cartridge (or perhaps you held down the SELECT key while booting, if you have an XL computer). If so, simply turn off the computer's power, insert the BASIC cartridge (if you don't have an XL computer), and turn the power back on again.

At this time, and for the rest of this section, we will presume that you are in Atari BASIC or BASIC XL. Any time the READY prompt appears, you may return to the DOS XL menu by simply typing the BASIC command, DOS. Generally, you may again return to BASIC without losing any program you may have in memory by again selecting the "T" menu option. But PLEASE be sure and read the cautionary note in section 3.13.

If you have chosen to use the DOS XL command processor mode ("CP"), you can enter the BASIC cartridge via the CAR command. If you then use BASIC's DOS command, you will be returned to CP instead of the menu.

NOTE that using Q from the menu (thus choosing the command processor--CP--of DOS XL) and then using the CAR command from CP will gain you over 2,000 bytes of user space as compared to using menu option "T". This is because BASIC is then allowed to use all the space formerly occupied by the MENU. See also section 8 for information on the DOS XL boot process.

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 XL or Atari BASIC only. That is, these commands should be typed immediately after the READY prompt or used with a line number within your BASIC program.

The commands are presented in alphabetical order.

4.1 CLOSE

command: CLOSE

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

users: BASIC XL and Atari BASIC users

usage: CLOSE #fn

argument: fn - file number 1-7

examples:
```
CLOSE #1
CLOSE #OUTFILE
```

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 #0, the keyboard/screen), and can be used to advantage for this purpose when desired.

4.2 ENTER
---

command: ENTER

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

users: BASIC XL and Atari BASIC users

usage: ENTER filespec

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

examples: ENTER "D:PROGR1.LIS"
ENTER OVERLAYFILE$

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"
30 PRINT  "AND NOW FINISHING"
LIST "D:PROG1"

NEW
10 PRINT  "WE ZAPPED THE OTHER LINE 10"
20 PRINT  "AND NOW PROGRAM 2"
LIST "D:PROG2"
```

(continued on next page)

(section 4.2, example, continued)

```
NEW
ENTER "D:PROG1"
LIST
[and the computer will LIST the following:
        10 PRINT  "THIS IS PROGRAM 1"
        30 PRINT  "AND NOW FINISHING" ]

ENTER "D:PROG2"          [do NOT type NEW]
RUN
[and the computer will respond with:
        WE ZAPPED THE OTHER LINE 10
        AND NOW PROGRAM 2
        AND NOW FINISHING    ]
```

Notice how the two programs have been neatly merged together and how line 10 from program 2 has replaced line 10 from program 1. Remember: like numbered lines from an ENTERed program replace lines in memory, but otherwise the program in memory (if any) is not changed.

4.3 GET

command: GET

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

users: BASIC XL and Atari BASIC users

usage: GET #fn,avar

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

examples: GET #1,BYTE
GET #INFILE,VALUE

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 generally 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
```

NOTE: BASIC XL users may specify channel zero (GET #0). Atari BASIC users are limited to file numbers 1 through 7.

4.4 INPUT

command: INPUT

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

users: BASIC XL and Atari BASIC users

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

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

examples: INPUT #3,NAME$
INPUT #INFILE,VALUE1,VALUE2

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 BASIC XL or 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.

EXAMPLE PROGRAM:

```
10 DIM LINE$(100) : REM a string for INPUT
20 OPEN #1,8,0,"D:TEST" : REM create test file
30 FOR I=1 TO 10 : PRINT #1; "RECORD #";I
40 NEXT I  :  REM we wrote 10 lines to the file
50 CLOSE #1 : REM close the file D:TEST
60 OPEN #1,4,0,"D:TEST" : REM ready to read it
70 INPUT #1,LINE$ : REM get a line from file
80 PRINT LINE$ : REM and show it on screen
90 GOTO 70 : REM and go get another line
```

Note that this program will STOP at line 60 with an error (number 136), indicating it has reached the end of the file. You could use TRAP to good effect here (see your reference manual).

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.

4.5 LIST

command: LIST

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

users: BASIC XL and 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

examples: LIST "D2:PROG.LIS"
LIST FILE$, 1000, 2000

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 DOS XL, 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 4.2 on the ENTER command.

4.6 LOAD
---

command: LOAD

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

users: BASIC XL and Atari BASIC users

usage: LOAD filespec

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

examples:
```
LOAD "D:GAME.SAV"
LOAD FILE$
```

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   ]
```

4.7 NOTE
---

command: NOTE

purpose: This command determines the current physical disk location of an OPEN file for later use with the POINT command.

users: BASIC XL and Atari BASIC users

usage: NOTE #fn, avar1, avar2

arguments:
- fn - file number from 1 to 7
- avar1 - a variable to receive the current sector number
- avar2 - a variable to receive the byte offsett within the current sector

examples:
```
NOTE #1, SECTOR, BYTE
NOTE #INFILE, S, B
```

description:

This command generally requires an in-depth understanding of BASIC and data files on the part of the programmer before it can be used properly.

"Version 2" of both DOS XL and Atari DOS maintains only sequential files, with a forward link and file number check occupying the last 3 bytes of each physical sector. Version 2 provides no direct random-access capabilities on a file level; and, without the use of NOTE and POINT, the programmer is restricted to reading and updating a file by starting only at its beginning.

However, thanks to NOTE and POINT and the fact that the forward link includes a file number check, the experienced programmer may create his/her own random access index into either an existing file or one being built.

NOTE simply notes the current disk sector and byte offset within that sector for any currently opened disk file. It places the sector and byte values into variables supplied by the programmer. It is the programmer's responsibility to retain and remember the NOTEd values until needed by the POINT statement.

The following example is not exhaustive, but it does give at least a start on understanding the implementation of random access files under version 2 of DOS XL or Atari DOS.

```
100 REM part 1: build a file on disk
110 DIM LINE$(150) : REM an arbitrary size
120 DIM SECTOR(100),BYTE(100) : REM ditto
130 OPEN #1,8,0,"D:TESTFILE" : REM a new file
140   FOR I=1 TO 100
150   PRINT "GIVE ME LINE ";I; : INPUT LINE$
160   IF LEN(LINE$)=0 THEN 200
170   NOTE #1,SECTOR,BYTE
180   SECTOR(I)=SECTOR : BYTE(I)=BYTE
190   PRINT #1;LINE$ : NEXT I

200 REM done creating the file
210 CLOSE #1
220 MAXREC = I-1  : REM really...check it out

250 REM part 2: accessing the file
260 OPEN #1,12,0,"D:TESTFILE" : REM file we made

300 REM the main loop
310 PRINT "GIVE ME A NUMBER FROM 1 TO ";MAXREC
320 PRINT "  (OR 0 TO QUIT) ";
330 INPUT RECORD
340 IF RECORD=0 THEN END
350 SECTOR=SECTOR(RECORD) : BYTE=BYTE(RECORD)
360 POINT #1, SECTOR, BYTE
370 INPUT #1, LINE$
380 PRINT "LINE ";RECORD;"==";LINE$
390 GOTO 300
```

If you type in and run this program, be aware of the following: When the program asks for a line, you may hit just RETURN and the entry phase of the program will terminate. When the program asks for a record number, no check is made to see if the number you give is a legal one (i.e., from 1 to MAXREC).

If you have difficulty following this example program, we would suggest ignoring the subject of random access files for now. Not all programs and programmers have need of such files, so it may not be advisable to spend too much time on this subject.

If you are a programmer experienced on machines where random access via byte relative position is possible, you may want to consider OS/A+ version 4, which is OSS's DOS for Atari computers and double density (or larger) disk drives. File positioning under version 4 is as simple as specifying a 24-bit byte number which is the offset from the beginning of the file. Version 4 is NOT recommended for beginners and/or for those who need compatibility with Atari DOS.

4.8 OPEN

command: OPEN

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

users: BASIC XL and Atari Basic users

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 0)
- filespec - a proper OS/A+ filename

examples:
```
OPEN #1,8,0,"D:NEWFILE"
INMODE = 4 : INFILE = 3
INPUT INFILE$
OPEN #INFILE,INMODE,0,INFILE$
```

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 4. The number 0 is reserved for the screen and can not be used in Atari BASIC (though it is allowed in BASIC XL). 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 DOS XL version 2 or under Atari DOS.

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 have been SAVEd simply as "D:name" instead of as "D:name.ext").

EXAMPLE:

```
100 OPEN #1,6,0,"D:*" : DIM LN$(40)
        (setup to read the directory)
110 FOR I = 1 TO 20 : INPUT #1, LN$
        (we allow for a maximum of 20 names)
120 IF LN$(2,2)=" " THEN PRINT I,LN$(3,10) : NEXT I
        (if the second character is not a blank,
         we just read the line "NNN FREE SECTORS"
         which appears as last line of directory)
130 CLOSE #1 : OPEN #1,6,0,"D:*"
        (setup to read the directory again)
140 PRINT : PRINT "WHAT PROGRAM TO RUN ";
150 INPUT J : IF J>=I THEN GOTO 140
        (if program number is too big, try again)
160 FOR I = 1 TO J : INPUT #1,LN$ : NEXT I
        (search for the program user wants)
170 CLOSE #1 : LN$(1,2) = "D:"
        (replace "* " or "  " with "D:")
180 RUN LN$(1,10)
        (remember, "D:filename" can't be
         longer than 10 characters)
```

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

4.9 POINT

---

command: POINT

purpose: This command requests a change of the current physical disk location of an OPEN file for later access by some I/O command or statement.

users: BASIC XL and Atari BASIC users

usage: POINT #fn, avar1, avar2

arguments:
- fn - file number from 1 to 7
- avar1 - a variable which specifies the desired sector number
- avar2 - a variable which specifies the byte offsett within the desired sector

examples:
```
POINT #1, SECTOR, BYTE
POINT #INFILE, S, B
```

description:

This command generally requires an in-depth understanding of BASIC and data files on the part of the programmer before it can be used properly.

"Version 2" of both DOS XL and Atari DOS maintains only sequential files, with a forward link and file number check occupying the last 3 bytes of each physical sector. Version 2 provides no direct random-access capabilities on a file level; and, without the use of NOTE and POINT, the programmer is restricted to reading and updating a file by starting only at its beginning.

However, thanks to NOTE and POINT and the fact that the forward link includes a file number check, the experienced programmer may create his/her own random access index into either an existing file or one being built.

POINT simply specifies a desired disk sector and byte offset within that sector for any currently opened disk file. If the sector specified is actually part of the disk file OPENed on the given channel, and if the byte offset is valid, then the next access to that file channel (either input or output) will take place starting at the requested location. Generally, POINT is only valid for most operations when the file has been OPENed for update (mode 12).

See section 4.7, NOTE, for an example program.

4.10 PRINT

---

command: PRINT

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

users: BASIC XL and Atari BASIC users

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.

examples:
```
PRINT #3,"hi there",1,2,3
PRINT #OUTFILE, NAME$
```

description:

When a file number is used with the PRINT command, the specified expressions 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. Normally, the semicolon should be used.

See INPUT (section 4.4) for more information and an example program.

4.11 PUT
---

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
aexp - an arithmetic expression

examples: PUT #3,65
PUT #OUTFILE,ASC("A")

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 GET (section 4.3) for a program example.

4.12 SAVE

---

command: SAVE

purpose:

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

users BASIC XL and Atari BASIC users

usage: SAVE filespec

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

examples:
```
SAVE "D2:GAME.SAV"
SAVE FILE$
```

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.

4.13 XIO

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: BASIC XL and Atari BASIC users, but see notes on individual subcommands

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

arguments: subcommand -- see descriptions in subsections which follow.

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 10 of this DOS XL 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.

NOTE: Although BASIC XL users may use XIO to perform this subcommand, BASIC XL provides an easier method of accomplishing the same function via its RENAME command. See your BASIC XL manual for further details.

---

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.

| NOTE: Although BASIC XL users may use XIO to perform this subcommand, BASIC XL provides an easier method of accomplishing the same function via its ERASE command. See your BASIC XL manual for further details. |

---

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.
```

NOTE: Although BASIC XL users may use XIO to perform this subcommand, BASIC XL provides an easier method of accomplishing the same function via its PROTECT command. See your BASIC XL manual for further details.

---

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).

| NOTE: Although BASIC XL users may use XIO to perform this subcommand, BASIC XL provides an easier method of accomplishing the same function via its UNPROTECT command. See your BASIC XL manual for further details. |

Section 5: INTRINSIC DOS XL COMMANDS

Instrinsic Commands are one class of commands which can be given anytime the D1: (or D2:, etc.) prompt appears from the DOS XL Command Processor.

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
@         - Start a batch file execution
Dn:       - Change default disk drive
```

and there are a few others.

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

```
D1:DIRECTORY    list all files of disk on drive one
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
```

Detailed explanations of all Intrinsic Commands follow, presented in alphabetical order.

5.1 @

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 9 and 7.6 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 may result if the command @GEORGE.EXC is tried.

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

5.2 CAR

command: CAR

purpose: This command transfers control to a cartridge

usage: CAR

arguments: none

Description

The CAR command allows the user to enter a cartridge from DOS XL. 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 DOS XL cannot be used with these cartridges.

If no cartridge is present, using this command will cause an error message to be given.

5.3 Dn:

command: D1: or D2: or D3: or D4:

purpose: This command changes the default disk drive designator.

usage: Dn:

users: Owners of more than one disk drive

Description

Whenever the console processor of DOS XL is ready to accept a command from you, it prompts you with "D1:". This prompt serves a secondary purpose. It reminds you what the current default disk drive designator is.

Anytime you specify a filename to CP (in either an intrinsic or extrinsic command), if you omit the disk specifier, CP prefixes the filename with the same three characters it prompts you with (e.g., "D1:"). Thus, PROTECT GEORGE.DAT is seen by the protect execution code as PRO D1:GEORGE.DAT, and it does not have to worry about whether or not you used a proper drive prefix.

If you have more than one drive, however, it is sometimes convenient to designate a drive other than drive one as the default disk drive. So, if you type "D2:" in response to the DOS XL prompt, CP will acquire a new prompt ("D2:") and any files given without a drive designator will be presumed to be on drive 2.

Note that DOS XL actually supports D1: through D8:, but standard drives may be only be addressed as D1: through D4:. Drives from some other manufacturers may possibly be able to use the additional designations.

CAUTION: As shipped, DOS XL can only access D1: and D2:. To allow access to other drives, see section 7.5 and Appendix A.

5.4 DIR

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

```
ABX.CXX
AB.CUR
ABCDEF.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.

CAUTION: Specifying a disk file name as an output file name will not generally work unless the output file is on a diskette other than the one given or implied by the first file specifier.

5.5 END

command: END

purpose: Stop batch execution from within an execute file

usage: END

arguments: none

Description

The END command causes DOS XL 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.

5.6 ERA

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.

5.7 LOA
----

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:

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).

5.8 NOS

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.

5.9 PRO

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.

EXAMPLES:

{D1:}PRO *.*
will protect all files on drive 1

{D1:}PRO D2:*.COM
will protect all files on drive 2 which have an extension of "COM"

5.1Ø REM
---

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.

5.11 REN
---

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 of both DOS XL and Atari DOS, 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.

EXAMPLES:

{D1:}RENAME TEST.DAT TEST.BAK
will rename the file "TEST.DAT" on drive one to "TEST.BAK"

{D1:}REN D2:DOSXL.SUP DOSXL.SYS
will rename the file "DOSXL.SUP" on drive two to "DOSXL.SYS"--note that the drive specifier was NOT given for the new filename

5.12 RUN

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 DOS XL to perform a jump to the indicated address (or to the address contained in the CP's 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 DOS XL 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.

5.13 SAV
---

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:

```
SAVE PAGE4000 4000 40FF
```

This example will save the 256-page of memory at $4000 to the disk file PAGE4000 on the current drive.

5.14 SCR
---

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.

5.15 TYP
---

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:
```

Another use of TYPe is to create a text file. As noted elsewhere in this manual, you can create a new STARTUP batch file via the following:

```
TYP E: STARTUP.EXC
```

When E: is the source "file", as in this example, you must use a CONTROL-3 (hold down the CTRL key and type a 3) to terminate the entry (this creates an end of file signal--always true from E: but not well documented by Atari). Also, you may not use the cursor control keys to edit any line for which you have already pressed the RETURN key. Thus this method is handy for small text files but not be used as a general file creator.

Finally, the TYPe command may also be used to copy TEXT files from one disk file to another by using disk file names for both the input and output files.

5.16 UNP
---

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.

EXAMPLES:

{D1:}UNP *.BAK
will remove the protection from all files on drive 1 which have an extension of "BAK"

{D1:}UNP D3:DOSXL.SUP
will remove the protection from only the file "DOSXL.SUP" on drive 3

Section 6: EXTRINSIC DOS XL COMMANDS

The extrinsic commands are programs that are run by the Command Processor (CP) of DOS XL. Any binary file containing the .COM extension may be used as a DOS XL extrinsic command. The DOS XL COPY command is one such extrinsic command. If you perform the DOS XL 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 DOS XL, 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 DOS XL, 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, DOS XL 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 DOS XL system master diskette (except that some commands may be specific to particular versions or packages).

6.1 C65

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 ***

*** YOU MUST PURCHASE C/65 SEPARATELY -- IT IS NOT SUPPLIED AS A STANDARD PART OF DOS XL ***

6.2 CLRDSK

command: CLRDSK

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

users: Non-Atari disk drive users

usage: CLRDSK

arguments: none

options: none

Description:

This utility 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.

NOTE: In general, do NOT use CLRDSK unless instructed to do so by your drive's manufacturer. If CLRDSK does not appear on your master disk, it is not necessary for proper operation of the drive(s) you have bought.

6.3 CONFIG

command: CONFIG

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

users: owners of configurable drives

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

arguments: an optional list of parameters 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.

NOTE: DOS XL version 2 can NOT access the second side of double sided disk drives ("DD"). Inquire about DOS XL version 4 if you have such drives.

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.

6.4 COPY
---

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 to copy from a single density diskette to a double diskette on a single drive. Instead, use SDCOPY (section 6.13).

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.

6.5 DO

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 DOS XL intrinsic and extrinsic commands, however. For example, the following DO command would activate the assembler/editor cartridge or OSS MAC/65, enter a source program, and assemble it:

```
DO CAR;ENTER "D:PROGRAM.ASM";ASM ,#P:,#D:OBJECT
```

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 9 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"
```

NOTE: some programs which inspect the DOS XL command line themselves cannot be used with DO. CP's COPY command is an example of such a program.

6.6 DUPDBL

command: DUPDBL

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

users: 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 DOS XL (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.

See also sections 7.3 and 7.4 for mixed density copies.

6.7 DUPDSK
---

command: DUPDSK

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

users: all EXCEPT those using double density

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 DOS XL. Unpredictable and disastrous results may occur! DO use DUPDBL (see previous section) for this purpose.

See also sections 7.3 and 7.4 for mixed density copies.

6.8 INIT
---

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 DOS XL 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.). When the initialization process is complete, the floppy disk may now be used to store data.

The INIT utility presents you with 4 options:

1. Format Disk Only
2. Format Disk and write DOS.SYS
3. Write DOS.SYS only
4. Exit to DOS XL

CAUTION: If using DOS XL with either a SuperCartridge or an Atari XL-series computer, you may not use options 2 or 3 to write DOS.SYS to the disk. You must use COPY to add both DOS.SYS and DOSXL.SYS if you wish to make a bootable DOS XL disk. (A disk with DOS.SYS alone will boot, but will not take advantage of the extra memory available in these configurations.) If the system was booted from a master disk which did not have the file DOSXL.SYS, then options 2 and 3 are safe to use. See also sections 3.7 and 7.7.

6.9 INITDBL

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: DOS XL 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 6.8).

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 will 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.

6.1Ø MAC65

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.

Section 6.10 (MAC65 continued)

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 ***

*** YOU MUST PURCHASE MAC/65 SEPARATELY -- IT IS NOT SUPPLIED AS A STANDARD PART OF DOS XL ***

6.11 MENU
---

command: MENU

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

usage: MENU

arguments: none

options: none

Description

Although we firmly believe that the command system of the OSS 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 MENU command which provides menu access to the most frequently used system commands.

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

Refer to Section 3 for details on using the DOS XL MENU.

6.12 RS232

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 DOS XL 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. See Appendix B for a possible fix to this problem.

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.

6.13 SDCOPY

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 [-FQRV]

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 DOS XL 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.

Examples:

```
SDCOPY *.* -Q
        will copy all the files on the current
        single density diskette to a double
        density diskette in the same drive
SDCOPY ABC DEF -R
        will copy double density file "ABC" to
        the single density diskette in this
        same drive. The copied file will be
        named "DEF" on the single density disk.
```

Section 7: MULTIPLE DRIVES, MULTIPLE DENSITIES

Much--but not all--of the material you have read up until now assumes or implies that you have only a single disk drive. While DOS XL can function perfectly well on a single drive, it really begins to show its power when you connect two or more disk drives to your Atari computer.

If you have only Atari 810 disk drives, very little needs to be said about using them to advantage. Generally, you will find that DOS XL will perform operations such as COPY (menu option C) and DUPDSK (menu option D) much faster if you specify that your "source" (the file or disk you are copying from) is on one drive and your "destination" (the diskette to receive the copied material) is on the other drive.

It does NOT matter which drive is the source and which is the destination. But be aware that many DOS XL commands require that a system master diskette be installed in drive 1 (or the "default drive" at least--see section 5.3 if you are not using the menu). This is NOT a limitation, since all utilities which need to be loaded from the system master either wait for you to give a response before executing (e.g., INIT) or allow you to specify that you want such a wait (e.g., the "-W" option of COPY). (And note that these wait options are always automatically chosen for you when you use a menu option.)

If you have a non-Atari drive, we would suggest you read the rest of this section.

Many of the procedures discussed in the subsections which follow require the use of commands and options which are NOT available from the "ordinary" DOS XL menu level. You, as the user, have two ways of using these commands. First, you may use the "X" (Xtended command) menu option. When you use "X", DOS XL prompts you for a command line. At that time, type in the command as shown in the descriptions below. Second, you may use the CP level of DOS XL (possibly by choosing option "Q" from the menu). Again, simply type in the command as shown below when you are prompted by "D1:".

Remember, when using the CP commands CONFIG, INITDBL, INIT, COPY, and SDCOPY, as described below, you MUST have a DOS XL master diskette in drive 1 when you type the command. If you do not, you will get a FILE NOT FOUND error message.

7.1 Setting Up Multiple Drives

DOS XL version 2 is compatible with and capable of controlling any mixture of up to eight single density and/or double density disk drives. If you have a drive capable of double density operation as well as one or more drives only capable of single density, we would suggest that you connect the double density drive as drive 1 (see your drive manufacturer's manual for switch settings, etc.). This will allow you to boot DOS XL in either single or double density mode.

If a drive is capable of either single or double density operation, you can generally predict what state it will be in when power is turned on. If it is drive 1, it will acquire the density of the booted master disk. If it is other than drive 1, it will adopt its density from the switch setting on the drive's controller (again, see your drive manufacturer's manual). Of course, if it is not capable of double density operation (e.g., an Atari 810 Disk Drive), it will always be single density.

DOS XL as shipped is set up to handle 1 or 2 disk drives and up to 3 simultaneously open files in double density mode (i.e., 3 BASIC "OPEN" statements without an intervening "CLOSE"). If you own 3 or more disk drives, or you require more files open at one time, you must change the values in certain system variables. See section 7.5 for more information.

DOS XL automatically asks each drive what density it is when the boot process occurs. From then on, if you want to change a drive's density, you must use the CONFIG command from DOS XL's Command Processor.

CONFIG has several options, and you may read section 6.3 of this manual for more information on its capabilities. For our purposes, however, we need to learn three of its abilities before going on. Remember, the following commands MUST be performed from CP or via the X menu option.

1. If you wish to find out what density DOS XL believes each drive is, wait until you are prompted as above.

   You type: CONFIG [RETURN]

   The table which is printed will tell you what density each disk drive is as well as how many sides DOS XL is accessing (always 1 side with version 2). If a drive is not capable of double density

operation, it is noted as "can't configure". Up to eight drives will be reported, and the system will even tell you which drives you don't have. CAUTION: the fact that CONFIG reports a drive as present does NOT imply that it is accessible to DOS. See section 7.5 for more on this subject.

2. If you wish to change a drive from single to double density, wait until you are prompted as above.

   You Type: CONFIG 1D [RETURN]
   or
   CONFIG 2D [RETURN]
   etc.

   Here, you specify the drive NUMBER you want to configure and then use a "D" to indicate Double density. Any configurable drive may be changed this way. The complete CONFIG table will be displayed.

3. If you wish to change a drive from double to single density, wait until you are prompted as above.

   You Type: CONFIG 1S [RETURN]
   or
   CONFIG 2S [RETURN]
   etc.

   Here, you specify the drive NUMBER you want to configure and then use an "S" to indicate Single density. Any configurable drive may be changed this way. The complete CONFIG table will be displayed.

| CAUTION: Whenever you duplicate diskettes using more than one drive, whether using the "D" menu option or DUPDSK or DUPDBL, you MUST be sure that both drives are CONFIGured to the same density! You may use CONFIG (as in 1., above) to check that both drives are the same density and/or change them to be the same. See sections 7.3 and 7.4, below, for information on cross-density transfers.

7.2 Initializing Other Densities

We noted in section 3.7 that you should only use the "I" menu option to initialize a disk which is the same density as the booted master disk. Using the extended options of DOS XL, however, there are several other possibilities. And there is one exception to 3.7, also.

1. If you have a single drive system, you may initialize a double density diskette even if you have booted a single density master. To do so, when you are prompted as above,

   You Type: INITDBL [RETURN]

   INITDBL is a command which, when it has loaded from the DOS XL master disk, will simply ask you which drive you wish to use (presumably 1, in this case). It will then automatically configure the drive to double density, format the diskette, write DOS.SYS to the diskette, and then reconfigure the drive back to single density. DO NOT use this command when you have booted a double density master disk. Use INIT or menu option "I" instead.

2. If you have a multiple drive system, you may use CONFIG to configure drive 2 (or 3 or any other drive) to the density you desire. You may then wait for appropriate prompt, and

   You Type: INIT [RETURN]

   and the INIT utility will load and run, presenting choices identical to those presented by the "I" menu option. (Or, at the menu prompt you may use menu option "I".) The actual rule, simplified in section 3.7, is that you may initialize any diskette to the density of the drive it is placed in. (And you may check or set the density of a drive via CONFIG.)

7.3 Copying Between Densities, Single Disk System

This section applies only to those with one drive and assumes that that one drive is a configurable (single/double density) drive. If you have two or more drives, see the next section.

If you would like to copy one or more files from a single density diskette to a double density diskette (or vice versa), you must first have a diskette which has been formatted (initialized) to the proper density. If you do not have such a diskette, we suggest that you boot a master disk of the proper density and use the "I" option to initialize a blank disk. (Or see section 7.2, above, for use of INITDBL.)

Then, after having used the "Q" option of the menu, wait for the appropriate prompt,

and You Type: SDCOPY *.* -Q [RETURN]
or
SDCOPY *.* -QR [RETURN]

The first form will copy files from single to double density. The second form will copy files from double to single density.

SDCOPY will load in and then allow you to place your source ("from") diskette (if it is not the master disk) in the drive. SDCOPY will then read the files directory of the source disk and give you a chance to say Yes or No about each file in the directory. If you answer Yes, the file will be copied to your destination ("to") disk. Since you have only a single drive, you will have to swap diskettes at least once for each file (long files may require two or three swaps).

SUGGESTIONS: (1) Read section 6.13 for other options available with SDCOPY. (2) Remove any cartridges in your machine if you are copying large files. They may copy in fewer swaps, since the cartridge space is used by SDCOPY. (This does not apply to OSS SuperCartridges, which already automatically release their space to DOS XL.)

7.4 Copying with Multiple Drives

If you own 2 drives, you may instead use the standard "Copy Files" command (option "C" of the menu) to transfer files between single and double density.

First, however, you must ensure that your drives are appropriately configured.

Since you presumably have booted your double density master diskette on drive 1, we suggest that you use "CONFIG 2S" (see 7.1, above) to place drive 2 in single density mode (unnecessary, of course, if drive 2 is an Atari 810 Disk Drive, since it is always single density).

Remember, to use COPY or menu option "C" after using CONFIG, you must wait for the appropriate prompt. When asked for "from" and "to" file names (or when placing them in the command line for COPY), be sure and specify "D1:" and "D2:", as appropriate. (If you make a mistake, Copy will probably not find the file names you are looking for, so no harm will be done. Just reverse the drive specifiers and try again.)

SUGGESTION: The COPY command available from the DOS XL Command Processor is extremely flexible and powerful. It may be in your best interests to learn its secrets even if you do not want to learn all about CP. (By the way, note that the menu command "C" invokes COPY with the -Q and -W options requested.)

CAUTION: You may NOT use menu option "D" (Duplicate disk) to copy from a double density to single density diskette or vice versa! Strange and disastrous things will occur if you attempt to do so. Similarly, you may not use the CP commands DUPDSK or DUPDBL for this purpose. Duplicate disks are literally duplicates, including the fact that the densities MUST be the same.

7.5 Using 3 or more drives

DOS XL as shipped is set up to handle 1 or 2 disk drives and up to 3 simultaneously open files in double density mode (i.e., 3 BASIC "OPEN" statements without an intervening "CLOSE"). If you own 3 or more disk drives, or you require more files open at one time, you must change the values in the system variables DRVBYT and SABYTE as follows:

Changing the number of drives: As shipped, the system variable DRVBYT, location 1802 decimal, holds the value 3, which allows you to use 1 or 2 disk drives. If you need to change this value, refer to the following table for new values:

value:	number of drives:
1	only 1
3	1 or 2
7	1, 2, or 3
15	1 to 4
31	1 to 5
63	1 to 6
127	1 to 7
255	up to 8

In order to change the value in DRVBYT, perform the following steps:

1) Insert the Atari BASIC cartridge into your computer (do NOT use BASIC XL). Boot your DOS XL master disk.
2) Ensure that the DOS XL menu is not active by issuing the "Quit to DOS XL" command.
3) Enter the BASIC cartridge via the CAR command.
4) POKE location 1802 with the desired value.
5) Hit the SYSTEM RESET key.
6) Issue the "DOS" command to return to DOS XL.
7) Type: MENU [RETURN] to get back into the menu.
8) Use the "Initialize Disk" command with option 3 to write DOS.SYS onto your master disk.

Changing the number of simultaneously open files: As shipped, you may open at the same time up to 6 single density files, or up to 3 double density files. If this setting is not enough for your application, you may change the value in the system variable SABYTE, location 1801 decimal, according to the following table:

value:	files open in single density:	files open in double density:
2	2	1
4	4	2
6	6	3
8	--	4
10	--	5
12	--	6

In order to change the value in SABYTE, perform the following steps:

1) Insert the Atari BASIC cartridge into your computer (do NOT use BASIC XL). Boot your DOS XL master disk.
2) Ensure that the DOS XL menu is not active by issuing the "Quit to DOS XL" command.
3) Enter the BASIC cartridge via the CAR command.
4) POKE location 1801 with the desired value.
5) Hit the SYSTEM RESET key.
6) Issue the "DOS" command to return to DOS XL.
7) Type: MENU [RETURN] to get back into the menu.
8) Use the "Initialize Disk" command with option 3 to write DOS.SYS onto your master disk.

NOTE: You many change both SABYTE and DRVBYT (via POKEs) at step 4, if desired, thus executing the 8 steps only once.

See also Appendix A for further information on buffer allocation, sizes, etc.

7.6 Booting up directly into a BASIC program

DOS XL is capable of booting directly into a BASIC program. In order to do so, you must perform a few simple operations, which are presented in step-by-step fashion below:

1) Boot a master diskette, entering either the menu or the command processor.

2) If you want the startup file on another disk, place that disk in the drive at this time. If it is not initialized, refer to either section 3.7 (menu) or 6.10 (CP) for instructions on how to initialize it.

3) From the appropriate prompt (see above), use

   TYPE E: STARTUP.EXC [RETURN]

   At this time, the screen will be blanked out and the cursor will appear in the top left hand corner of the screen.

4) At this time, type the line

   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 this startup mode.

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. (To the Atari Computer's OS, this signals and end-of-file.) When this step has been executed, the file STARTUP.EXC will actually be written to the disk and control will return to the operating system and the menu or D1: prompt. (For information on the workings of EXC files in general and STARTUP.EXC in particular, see Sections 8 and 9.)

6) In answer to the D1: prompt type DIR (or use the "F" menu option) to obtain a directory of the disk. CAUTION: 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

If DO.COM is missing, us COPY (C menu option) to move it from your system master diskette to this disk. If your BASIC program is missing, SAVE it (from BASIC) to this disk.

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 its proper slot.

7.7 Making a Double Density Master Diskette

This section presumes that you were shipped a DOS XL on a single density diskette, only. Some disk drive manufacturers are now shipping either two disks or a flip-over disk with a copy of DOS XL in both single and double density. If you received such a disk, you do NOT need this section. Instead, simply boot the density desired and use menu option "I" (to initialize a disk of the appropriate density) or "D" (to duplicate the master diskette in the appropriate density). If you use option D, be sure to answer the density question prompt correctly. (Of course, you can also use DUPDSK or DUPDBL commands from the DOS XL CP.)

We provide here step-by-step instructions for both menu mode and CP mode. In either case, we assume you have booted a single density master diskette.

CAUTION: you should NOT use your original master disk for the procedure we are about to describe. Be sure and use a duplicated copy of your master, instead. Since we may rename a file, you may NOT have a write protect tab on the disk. This is VERY dangerous, hence the need for using only a copy of your master disk and NEVER the original.

From the DOS XL Menu:

1. Use option F. Inspect the files directory. If the filename DOSXL.SYS appears, use the U option, giving DOSXL.SYS as the filespec, and then use the R option. To the "Old name" prompt, answer DOSXL.SYS and to the "New name" prompt, answer simply DOSXL. If you did need to do this rename, reboot your system at this time (turn your computer power off and then on).

2. Use option X. To the "command:" prompt, answer

   INITDBL [RETURN]

   When you are prompted with "DRIVE TO INITIALIZE", answer with the numeral 1 followed by [RETURN]. The program will then prompt you to "INSERT DISK AND HIT RETURN". At this time, remove your system master diskette and insert a blank diskette. Then, and only then, hit [RETURN].

INITDBL takes a minute or so to complete its work. The usual "HIT RETURN FOR MENU" prompt will finally occur, and you should reinsert your master diskette before hitting [RETURN].

3. Use option X. To the "command:" prompt, answer

    SDCOPY *.* -Q

    The SDCOPY program will load and execute. When it asks you to insert the disk to be copied, do nothing except hit [RETURN] since you will be copying this system master diskette.

    SDCOPY will tell you which file you are about to copy and ask whether you wish to do so. If you hit Y [RETURN] the file will be copied. If you hit N [RETURN], it will not be. Presuming that you wish a complete double density master, the only file you do NOT wish to copy is DOS.SYS (since INITDBL has already written it to your new double density diskette). Answer N to DOS.SYS but Y to all other files. (If you KNOW you will not be using certain programs or files, you may answer N for them, also, but we would suggest making at least one full double density master.)

    SDCOPY is a slow and painful process of inserting and removing diskettes, but it will finally finish (and it is worth it).

4. If you used option R in step 1, above, use option R. We will now give DOSXL as the "old name" and DOSXL.SYS as the "new name", thus restoring the system master to its original condition.

5. Turn off the power to the computer. Insert your new double density diskette into your drive. Turn on the computer's power. DOSXL should boot in double density mode. By obtaining a diskette directory (option F), you can note whether DOS.SYS is now only 23 sectors long. If so, you have been successful.

6. If you used option R in step 1, above, use option R now. Again, we will rename from "old name" DOSXL to "new name" DOSXL.SYS, thus allowing extended memory operation the next time this disk is booted.

7. Label and write protect your new double density master.

From the DOS XL Command Processor prompt (D1:)
---

1. Use the DIR command. If the file DOSXL.SYS appears in the file listing, UNProtect it and then RENname it to simply DOSXL (with no extension). If you renamed DOSXL.SYS, reboot your system.

2. Use the INITDBL command to initialize a blank diskette in double density mode. Be sure to re-insert your master diskette when INITDBL is finished.

3. Use the command

   SDCOPY *.* -Q

   to copy all files except DOS.SYS from your single density master to your new double density master. Be sure to re-insert your single density master when SDCOPY is finished.

4. If you RENamed DOSXL in step 1, use RENname again to change its name back to DOSXL.SYS.

5. Insert your new double density master diskette and re-boot the computer. Check to be sure you do, indeed, have a double density master by using the DIR command. A double density DOS.SYS file is only 23 sectors long.

6. If you RENamed DOSXL in step 1, use RENname again to change its name on this double density diskette.

7. Be sure to label and write protect your new double density master diskette.

Section 8: THE DOS XL BOOT PROCESS

The process of loading the DOS XL operating system into your Atari's memory is somewhat different than the process for loading other DOS's. Also, deleting or adding certain files to a bootable disk can affect what portions of DOS XL are loaded. In order for you to modify this process and thereby customize your system, this section describes the steps which are followed in the boot process.

8.1 Extended Memory DOS Systems and DOSXL.SYS

As shipped, your DOS XL master diskette contains two special files. One is called "DOSXL.XL" and the other is called "DOSXL.SUP". We shall call these two files, collectively, the "extended memory DOS system(s)".

In order to take advantage of an extended memory DOS system, you MUST have one (or both) of the following:

1. An Atari XL-series computer with 64K Bytes of RAM (1200XL, 800XL, expanded 600XL, etc.)
2. An OSS SuperCartridge (ACTION!, BASIC XL, MAC/65, etc.)

If you have neither of these capabilities, please skip to section 8.2.

Again, as shipped, these extended memory DOS systems are NOT active. If you wish to take advantage of possible extended memory configurations on your computer, you should read the rest of this section. Otherwise, you may skip to section 8.2.

If you are using an OSS SuperCartridge for most of your work, you should rename DOSXL.SUP, following the procedure outlined below. If you are not using SuperCartridge but you are using an XL-series computer, you should rename DOSXL.XL, again using the following process:

If you are using the DOS XL MENU, choose option U. The "filespec" to be unprotected is either DOSXL.SUP or DOSXL.XL, depending on your system configuration as outlined above. Again, from the menu, choose option R. In response to the "Old name" prompt, answer either DOSXL.SUP or DOSXL.XL, as you did with the option U prompt. In response to the "New name" prompt, answer DOSXL.SYS and return to the menu.

If you are using CP, you should UNProtect DOSXL.XL or DOSXL.SUP, as noted above, and then RENname that same file to DOSXL.SYS before proceeding.

If you now reboot your system (turn your computer's power off and back on), an extended memory DOS system will be booted.

How It Works

While most DOS's reside only in the DOS.SYS file on a bootable disk, DOS XL can actually occupy two separate files. The first file, DOS.SYS must be on any disk to make it bootable. At the beginning of the boot process, this file is loaded into memory, occupying locations $700 to $1E00.

At that time, this DOS (it is actually a complete DOS in itself) checks to see if the file named DOSXL.SYS is on the booted diskette. If so, DOS.SYS presumes that DOSXL.SYS contains an extended memory DOS and loads it for you. Once DOSXL gets control, several things happen.

First, DOSXL checks to see if you do, indeed, have the memory configuration that you "claimed" to have when you renamed one of the DOSXL files. If you do not actually have such a system, DOSXL returns control to the original DOS.SYS and nothing more happens. For all intents and purposes, DOSXL.SYS is not active at alll in this circumstance.

If, however, your memory configuration is as you "claimed", DOSXL moves itself into the RAM memory "under" either the SuperCartridge or Atari's OS (as appropriate).

This newly loaded code now becomes the DOS of the machine. This DOS saves the user 3K Bytes to 5K Bytes of memory by occupying memory which is bank-switched with the SuperCartridge (by taking advantage of special hardware within the cartridge) or the Atari OS (again, by taking advantage of special hardware built into Atari XL-series computers).

Remember, then, if you desire NOT to load this special DOS file, DOSXL.SYS, simply rename the file to a name other than DOSXL.SYS (we recommend simply DOSXL, with no extension).

8.2 The AUTORUN.SYS file

During the boot process, and once the DOSXL.SYS file is either loaded, skipped, or not used at all (see previous section), DOS XL searches the disk for a file called AUTORUN.SYS (note that there is no such file on the DOS XL master disk). If this file is found, it is loaded into memory just as if you had issued a "Load Binary" menu command.

For example, one way to insure that the RS232 driver is loaded into memory each time you boot a certain disk is to rename the file "RS232.COM" to the name "AUTORUN.SYS" (see also the section on the file "RS232FIX.COM").

This loading of AUTORUN.SYS is compatible with the Atari DOS mode of operation, so most AUTORUN.SYS files which ran with Atari DOS will also run with DOS XL.

8.3 The STARTUP.EXC file

Again, during the boot process, there is yet another possible step.

If the file AUTORUN.SYS is not found, or if it returns to DOS with a 65Ø2 RTS instruction, DOS XL continues the boot process by searching for the file STARTUP.EXC. This file is a text file which contains commands to the DOS XL command processor.

On your DOS XL master disk there is a STARTUP.EXC file which contains REM commands for just putting messages to the screen, and the command MENU, which loaded and started the DOS XL menu (see the following section for another method of loading the menu).

SIDELIGHT: In order to change the contents of this file, just use the "Copy Files" option of the DOS XL menu and select "E:" and "D:STARTUP.EXC" as the "From" and "To" files, respectively. When the screen clears and the cursor appears at the upper left of the screen, type the desired commands, one to a line. When you are finished, type control-3 (hold down the control key and press 3). The commands you typed will then be written out to the disk into the STARTUP.EXC file.

If you desire not to have a STARTUP.EXC file, simply erase it or rename it to a different name (perhaps STARTUP.TXT, for "text file").

Certain cartridge-based products, including ATARI WRITER from Atari, Inc., will not work properly if your boot disk contains a STARTUP.EXC file. If you are using a product such as ATARI WRITER, make a special boot disk as follows: 1) Duplicate your master disk onto a blank one. 2) Erase the file STARTUP.EXC on that disk. 3) Erase the file MENU.COM on that disk. (only if you want more memory space) You should now use this disk for booting into ATARI WRITER (you may use this disk for booting into other products, but you will not have the menu if you go to DOS).

8.4 The MENU.COM file

As the last step of the DOS XL boot process, and presuming that neither AUTORUN.SYS nor STARTUP.EXC has taken control of the system, one further action may be performed.

The final step of the boot process is the loading of the DOS XL menu. DOS XL will search the disk for the file MENU.COM. If that file is found, it is loaded into memory in the lowest available address (the current value of the MEMLO pointer, locations $2E7 and $2E8) and will be in control at the end of the boot process. If the file MENU.COM is not found, the DOS XL command process will be in control.

At this point, if there is a cartridge inserted, it will be entered. Otherwise, the DOS XL menu or the DOS XL command processor, depending on which has control of the system, will be entered.

Section 9: Batch Processing

9.1 An Overview of Batch Processing

You may often find yourself repeating the same group of commands over and over. DOS XL 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 ACTION! programs that had to be run in sequence. You could do this in two ways:

1. Issue the CP extrinsic command for each program one at a time. If the running time of the programs was very long you might sit at the key board for hours just to type a program name every once in awhile.

OR

2. Create a BATCH file containing the DOS XL commands required to run the set of 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 a DOS XL 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 DOS XL to read it line by line, executing the CP commands just as if they were being entered from the keyboard.

9.2 .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 DOS XL Console Processor commands
2) they must be shorter than 128 characters in length
3) they must end in a carriage return (ATASCII $9B).

DOS XL 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 DOS XL. The CP 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.)

9.3 Intrinsic Commands for .EXC Files

DOS XL 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 DOS XL to keyboard entry mode (the CP).

See Section 5 for more detailed explanations.

9.4 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.

9.4.1 Stops by DOS XL

Humans are not quite perfect in the eyes of computers and sometimes make mistakes. DOS XL commands specified in error will generate error messages. If DOS XL 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 DOS XL, not just because a program generates an error.

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

Finally, execution of course stops when the end of the execute file is reached.

9.4.2 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 DOS XL. If a program sets this byte to zero, then upon returning to DOS XL the execute file execution will immediately stop. The execute flag is located 12 bytes from the start of DOS XL, which is pointed to by memory location 10 ($0A). The following BASIC program segment will turn off the execute file and return to DOS XL.

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

Or, from BASIC XL, you could simply use

```
100 Poke Dpeek(10)+11,0 : Dos
```

(Remember, though, that a CAR command automatically stops EXC file execution, so this example may not be useful from BASIC.)

9.5 STARTUP.EXC: A Special File

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

See section 8.3 for other details on STARTUP.EXC.

9.6 How Execute Files Work

When you type in the command "@filename", CP actually stores that filename in an internal buffer (CPEXFN) and sets a flag (CPEXFL) to indicate that a batch operation is in progress.

Each time CP prompts the user (e.g., with D1:), it checks this flag to see if batch is active. If so, it opens the batch file (using the stored filename). Unless this is the first time the batch file has been opened (again, kept track of via a bit in CPEXFL), CP POINTs to the start of the next text line in the file.

The next text line is then read into the command buffer. Then CP NOTEs the new position in the file and saves the position (in internal variable CPEXNP) for use by the next needed POINT process (as above).

Finally, the command in the command buffer is executed just as if the user had typed it from the keyboard.

If the command properly terminates (e.g., via an RTS or a JMP through DOSVEC), the entire process repeats, until the execute flag is somehow turned off.

The experienced programmer will no doubt realize that changing the contents of the various CPEXxx locations can affect batch execution in possibly very interesting ways. These locations are all defined in the file called SYSEQU.ASM and are offsets from the address contained in DOSVEC (location $000A). See, as an example, the "program controlled stop" mentioned in 9.4.2, above.

See also section 10.2.3, which repeats some of this material.

Section 10: Assembly Language and DOS XL

As mentioned in Section 1.6, DOS XL is designed as a layered operating system. Application programs (including languages such as BASIC XL) 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 DOS XL to another. (Probably the only other areas of change would involve memory map usage.)

In any case, herewith is a description of the proper assembly language calling sequences and parameters under DOS XL.

10.1 Interfacing to I/O Routines

10.1.1 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 6502's X-register. Thus, to access IOCB number four, the X-register should contain $40 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 $0340 to $03BF 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 10-1 gives the DOS XL 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 10-1

IOCB STRUCTURE

FIELD NAME	OFFSET WITHIN IOCB (bytes)	SIZE OF FIELD (bytes)	PURPOSE OF FIELD
ICHID	0	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 6502 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 10-1
IOCB STRUCTURE

IOCB field name	Ø	1	2	3	4	5	6	7
					BUFFER		PUT-A-	
	ICHID	ICDNO	ICCOM	ICSTA	ADDRESS		BYTE ADDRESS	
Type of command					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								
REName		*	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 10-2 IOCB Field Usage

8 / BUFFER LENGTH / ICBLEN	9	10 / ICAX1	11 / ICAX2	12 / ICAX3	13 / ICAX4	14 / ICAX5	15 / ICAX6	IOCB field name (as given in SYSEQU.ASM) / COMMAND NAMES
		mode						COPN
								CCLOSE
								CSTAT
length								CGTXTR
length								CPTXTR
length								CGBINR
length								CPBINR
(See section 10.1.2)								
								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 10.1.2
'byte' Byte in sector, see section 10.1.2

Figure 10-2 (con't.)

10.1.2 The I/O Commands

Figure 10-2 provides a table of I/O commands and their usage of the various fields of the IOCB's. The first seven are DOS XL 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 DOS XL 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 DOS XL and Atari DOS:

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 DOS XL is a TRUE operating system. The structure of device drivers is material for a later section (10.3), 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 DOS XL Commands

The OS itself only understands a few fundamental commands, but DOS XL also provides for the 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
6	Read Directory Only
8	Write Only
9	Write Only Append
12	Read/Write(Update)

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 10-2 shows several DOS XL system commands not yet discussed. These "extended" commands are accessed via the extended request routine in a device driver's handler table (see section 10.3 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 DOS XL 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.

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 DOS XL 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.

10.1.3 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 $02 to $7F (2 through 127 decimal) are presumed to be "warnings". Those from $80 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 $80-$FF, it must be an error
```

In theory, DOS XL 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.

10.2 Manipulation of DOS XL

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 DOS XL. See section 10.1 for further information about the OS interface.

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

10.2.1 SYSEQU.ASM

Every DOS XL master disk contains an assembler source file, SYSEQU.ASM, that has various commonly used Atari OS and DOS XL 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.

10.2.2 CP MEMORY LOCATIONS

The Command Processor (CP) on the Atari is designed to be placed just after the normal Atari File Manager when the DOS.SYS version of DOS XL is used. Since the actual location of CP may vary with different versions of the file manager and/or because of different memory configurations, a fixed location has been assigned to point to CP. The location CPALOC ($0A on the Atari) contains the address of the DOS XL and CP warmstart entry point. Most Atari programs should return to CP by JMPing to the address contained in CPALOC.

10.2.3 EXECUTE PARAMETERS

The CP execute flag is located CPEXFL ($0B) from the start of CP. 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	$80	If one, an execute is in progress
EXCSCR	$40	If one, do not echo execute input to screen
EXCSUP	$20	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	$10	If one, a new execute is starting. Tells CP to start with the first line of the file

CP 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, CP must save the execute file name and the three byte NOTE values. The filename is saved at CPEXFN ($0C) into CP. The three NOTE values are saved at CPEXNP ($1C) into CP. (CPEXNP = ICAUX5; CPEXNP + 1= ICAUX4; CPEXNP + 2 = ICAUX3). By changing the various execute control parameters, a programmer can cause chaining of execute files, skipping of certain lines in the file, etc.

10.2.4 DEFAULT DRIVE LOCATION

The CP default drive file spec is located at CPDFDV ($07) into OS/A+. The Default Drive here is ATASCII Dn: where "n" is the ATASCII default drive number.

10.2.5 EXTRINSIC PARAMETERS

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

```
D1:COPY  FROMFILE  D2:TOFILE
```

is an example of this. The entire command line is saved in the CP input buffer located at CPCMDB ($3F) bytes into CP and is available to the user. Since most command parameters are file names, CP provides a means of extracting these parameters as filenames. The routine that performs this service begins at CPGNFN ($03) bytes into CP . The routine will get the next parameter and move it to the filename buffer at CPFNAM ($21) bytes in CP. If the parameter does not contain a device prefix, then CP 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 ($0A into CP) 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     #0              ;OF JUMP
        STA     GETFN+2         ;INSTRUCTION
GETFN   JMP     0
```

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
```

10.2.6 RUNLOC

Whenever an Extrinsic command is invoked, RUNLOC ($3D into CP) 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 of your 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 CP. If you want to forbid re-entry, you need to set RUNLOC's high order byte ($3E into CP) to zero:

```
LDY     #RUNLOC+1       ;FORBID RE-ENTRY
LDA     #0              ;TO ME
STA     (CPALOC),Y
```

10.3 DEVICE HANDLERS

As we have noted before, CIO is actually a very small program (approximately 700 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, CIO 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.

10.3.1 The Device Handler Table

At location "HATABS" in RAM, CIO 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 DOS XL 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:". DOS XL 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 computers:

```
       *=      $031A
   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   0        ; zero marks the end of the
                         table
       .WORD   0        ; ...but there's room for
                        several
       .BYTE   0        ; ...more devices
                        et cetera
```

10.3.2 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 Atari's OS 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 10.3).

```
    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 10-3

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: CIO 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 DOS XL 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 DOS XL 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.

10.3.3 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. 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 the Atari computers which produces intrinsically 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.

10.3.4 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 XL 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.

Section 11: FILE STRUCTURE

DOS XL version 2 was produced to provide the maximum compatibility possible with Atari's DOS 2.0s. 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 720 data sectors available on an 810 (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 (810 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].

11.1 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.

11.2 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 11-1. 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
0	1	Flag byte. Meanings of bits: $00 Entry never used $80 Entry was deleted $40 Entry in use $20 Entry protected $02 a version 2 file $01 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 11-1
Directory Entry Structure

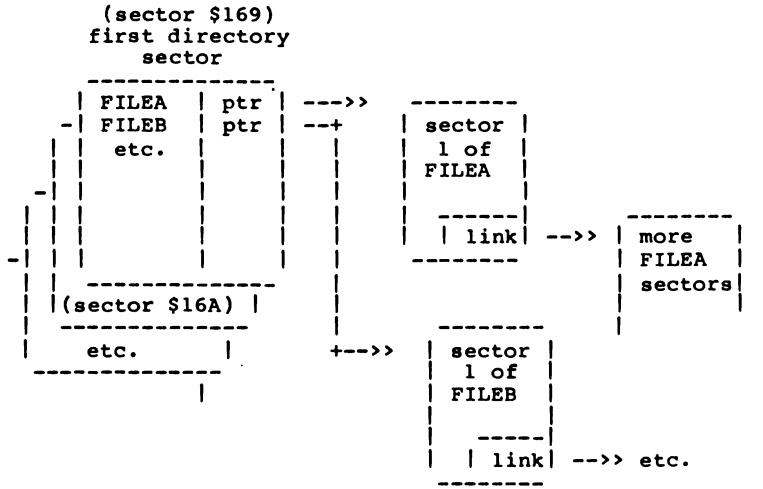


Figure 11-2

Version 2 Directory Structure

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

11.3 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 11-3 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 11-3

Structure of the VTOC Sector

Appendix A: CUSTOMIZING DOS XL

Although DOS XL 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 DOS XL to suit his own tastes. The following sections describe the most useful modifications which may be performed.

A.1 BUFFER ALLOCATION

DOS XL allows 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 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 DOS XL disk, to be sure of the latest system value. As of the printing of this manual, the following locations are in use:

label	location	use
SASA	$Ø7ØC	start of buffers
SABYTE	$Ø7Ø9	# 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:

Amount of space required: For single density diskettes, use 2 buffers 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.

Be sure you have enough room at the location you will specify by SASA to contain the buffers required.

Specifying the number of buffers: Do NOT include the drive buffers in the count of buffers you give DOS XL. Instead, simply specify the FILE buffers in location SABYTE.

A.2 SPECIFYING EXISTING DRIVES

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 DOS XL 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 DOS XL 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). However, this option can NOT be taken when an extended memory DOS system is active. See sections 3.7, 6.8, 7.7 for more information.

Appendix B: DOS XL AND THE 850 INTERFACE MODULE

B.1 Loading the RS232 handler

When using Atari DOS 2.0s, the only way to load the RS232 device handler (Rn:) from the 850 interface module is through the use of an AUTORUN.SYS file (see section 8.2). This option is also available to DOS XL users. Another option is, however, available to you. After booting DOS XL, you can simply issue the following commands:

1) from the DOS XL menu:
   You type: X
   and then, when prompted for a command,
   You type: RS232 [RETURN]

2) or, from the DOS XL command processor:
   You type: RS232 [RETURN]

This sequence of commands will cause the RS232 device handler to be loaded into the system. You can then refer to the 4 RS232 ports on the 850 interface module as "R1:" through "R4:", respectively.

B.2 Bugs in the RS232 driver

Unfortunately, the device handler which loads in from the 850 interface module is not perfect. The most serious flaw occurs when you push SYSTEM RESET after the RS232 driver has been loaded into memory. Under certain circumstances, your Atari computer will "hang", freezing the keyboard, after pressing that key. For this reason, many Atari reference books recommend that you NEVER press SYSTEM RESET after loading the RS232 driver. Under DOS XL, however, there is a solution to this and other problems. On your master disk there is a file called "RS232FIX.COM". This file is almost identical to the "RS232.COM" file which is normally employed to install the RS232 handler. The fixed version attempts to correct some of the known bugs in that handler.

You may ask, "Why not just include the correct version on the DOS XL master disk?" Well, Atari has produced several versions of the 850 interface module. OSS has almost no way of knowing whether the corrected version works with all such revisions so, rather than

introducing new problems, both the original and the fixed version are included.
To test the "RS232FIX.COM" file with your 850 module, either:

1) Using the DOS XL menu:
        You type: X

        When prompted for a command,
        You type: RS232FIX [RETURN]

2) or using the DOS XL command processor:
        You type: RS232FIX [RETURN]

If the RS232 handler loaded in this way seems to work properly, you may use it exclusively for loading the RS232 handler, ignoring the original RS232 command.

APPENDIX C - SYSTEM MEMORY MAPS

C.1 ATARI ZERO PAGE MAP

location		usage
0-9		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
80-FF		user and language zero page
80-CD		BASIC XL and Atari BASIC zero page
D2-FF		floating point zero page

C.2 ATARI SYSTEM MEMORY MAP - DOS XL version 2

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-varies		DOS XL -- file manager and CP or just buffers, etc., when using extended memory DOS systems
709	SABYTE	number of 128 byte file buffers
70A	DRVBYT	bit map: accessible drives
70C	SASA	address of start of buffers
(2E7)-BFFF		user, language, and graphics memory Note: (2E7) means "contents of location $02E7" (LOMEM).
A000-BFFF		SuperCartridge and Atari BASIC memory-- also used by DOS XL for file manager and CP in "DOSXL.SUP" version of the extended memory DOS system
C000-CFFF		Unused in Atari 400/800, OS ROM in XL-series, bank switched with RAM
D000-D7FF		I/O locations
D500-D5FF		Used by SuperCartridge for bank select
D800-DFFF		Floating Point ROM
E000-E3FF		Character Set ROM
E400-FFFF		OS Drivers, CIO, etc.--in ROM
E400-FFF9		Bank-selectable RAM used by DOS XL for file manager, CP, etc., in the "DOSXL.XL" version of extended memory DOS

Appendix D: Atari Writer and Other Cartridges

Certain cartridge-based products, including ATARI WRITER from Atari, Inc., will not work properly if your boot disk contains a STARTUP.EXC file. If you are using a product such as ATARI WRITER, make a special boot disk as follows:

1) Duplicate your master disk onto a blank one.
2) Erase the file STARTUP.EXC on that disk.
3) Erase the file MENU.COM on that disk.
   (only if you want more memory space)

You should now use this disk for booting into ATARI WRITER (you may use this disk for booting into other products, but you will not have the menu if you go to DOS).

APPENDIX E: Errors

E.1 TYPES OF ERRORS

All DOS XL operations return a status value in the IOSTAT field. DOS XL convention is that status values of $80 or greater indicate some sort of error. There are four fundamental kinds of errors that can occur with DOS XL:

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.

E.2 ERROR CODE LISTING

The list of error codes which follows is not necessarily exhaustive, but it does represent all error codes which will normally be returned fro DOS XL or any of the Atari device drivers.

ERROR CODE HEX	DECIMAL	MEANING
$01	1	No error or warning.
$02	2	Truncated ASCII line. The OS did not find a CR within BUFLEN for ASCII line I/O.
$03	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.
$80	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	130	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 $00, $10, $20, $30, $40, $50, $60, and $70 are valid. From some languages, these will be seen as channels 0 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 $A0 through $AF are file manager errors.

$A0	160	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	170	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.

a reference manual for

B U G / 6 5

an Assebly Language Debugging program for
use with 6502-based computers built by
Apple Computer, Inc., and Atari, Inc.

Rev 1.1

PREFACE

BUG/65 is an interactive debugging tool for use in the development of assembly language programs for the ATARI 800 or ATARI 400 personal computers. It's designed to take as much of the drudgery out of assembly language debugging as possible. The design philosophy behind BUG/65 is that the computer should serve as a tool in the debugging process as opposed to a hindrance. One result of this philosophy is that BUG/65 requires a relatively large amount of memory when compared to simpler debug monitors. This is the result of a tradeoff between memory and functionality, with function winning out.

BUG/65 is a RAM loaded machine language program occupying 8K of memory; it is self relocatable as shipped and requires a full 48K bytes of memory. BUG/65 is also designed to be floppy disk based - it isn't intended to be used in cassette-only systems. BUG/65 was designed for use by an experienced assembly language programmer.

BUG/65 is an original product of the McStuff Company, which developed the product under the name "McBUG", which name is their trademark.

For use on the ATARI 800 or 400 computer with a minimum of 48K of RAM and one floppy disk drive.

TRADEMARKS

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

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

Apple, Apple II, and Apple Computer(s) are trademarks of Apple Computer, Inc., Cupertino, CA

Atari, Atari 400, Atari 800, Atari Home Computers, and Atari 850 Interface Module are trademarks of Atari, Inc., Sunnyvale, CA.

TABLE OF CONTENTS

(continued)

TABLE OF CONTENTS (continued)

SUMMARY OF MAJOR FEATURES OF BUG/65

* A full set of debugging commands - change memory, display memory, goto user program with break points, etc.

* Binary file read and write, including appended write.

* A disassembler.

* An instant assembler providing labeling capability.

* Expanded command addressing capability: hex or decimal addresses, + and - operators supported, relocated addresses supported.

* Read or write disk sector(s).

* Multiple commands permitted in a command line. Command lines can be repeated with a single keystroke or repeated forever with the special slash operator.

* Support for relocatable assemblers - the base of a module can be specified and then used to reference addresses in that module.

* BUG/65 commands can be executed from a command file, and there is a command to create command files.

* Hex to decimal and decimal to hex conversions provided.

* Memory protection of BUG/65's code and data. BUG/65 won't allow you to use a BUG/65 command that will destroy any part of BUG/65 itself. For example, you can't use the Fill command to overwrite BUG/65's code.

* Page zero sharing. BUG/65 shares page zero with a user program by keeping two copies of the shared page zero locations - one for the user and one for BUG/65 itself.

SECTION 1 : COMMAND SUMMARY

This section is intended to be a handy reference guide and will probably prove indispensable after the user has thoroughly read through the rest of this manual. For the experienced debug user, might we suggest at least a quick perusal of Sections 2 through 6 and Sections 8 and 9.

The following table is simply a syntax summary of the available commands. Excepting for the first three commands (which are described in Section 8), all the commands are described in alphabetical order in Section 7.

COMMAND CODE	SYNTAX	PURPOSE
{RETURN}		Repeat last command line
/		When appended to a command line: repeat line forever.
=		Display last command line
A	A <addr>[/]	Ascii mode memory change
B	B <addr>	Base address for relocation
C	C <startaddr1> <endaddr1> <startaddr2>	Compare memory blocks
D	D <startaddr> [<endaddr>]	Display memory
E	E #filespec	Execute a command file
F	F <startaddr> <endaddr> [<value>]	Fill memory block with value
G	G [<startaddr>] [@<breakpoint> [Rn=<value>] [I=<count>]]	Go at address, set optional breakpoint, with optional Register value breakpoint and pass Counter.
H	H <number1> <number2>	Hexadecimal arithmetic result
I	I	disk Inventory (directory listing)
J	J #filespec,string	create command file
K	K <number>	convert hex to decimal

L	L <startaddr> <endaddr> <byte1> [<byteN> ...]	Locate byte string in memory block
M	M <startaddr> <endaddr> <toaddr>	Move memory block
P	P [S] [P]	Print output on Screen and/or Printer
Q	Q	Quit...go to OS/A+
R	R [<offset>] #filespec	Read a binary file to memory with optional offset
R%	R% [<sectornumber> [<bufferaddr> [<numsectors>]]]	Read sector(s) from disk to memory buffer
S	S <addr>↵	Substitute memory, numeric mode
T	T [S] [<count>]	Trace, with optional Skip over subroutine calls, for (optional) count instructions
U	U <addr> [<param>]	call User routine at given address and pass optional parameter in X,Y registers
V	V	View user registers
W	W [:A] <startaddr> <endaddr> #filespec	Write a block of memory to a binary image file, optionally appending instead of creating new file.
W%	W% [<sectornumber> [<bufferaddr> [<numsectors>]]]	Write sectors from memory buffer to disk
X	XA or XX or XY or XS or XP or XF	change user register value
Y	Y <startaddr> [<endaddr>]	disassemble memory block
Z	Z <addr>↵	instant assembler (at address)

SECTION 2: Notations Used In This Manual

The following notations are used in this manual:

<...> Is used to indicate a numerical address parameter. The address expression between the two characters "<" and ">" may be any valid address as described in Section 3. For example, <START> means that you can enter any valid address expression to specify the START parameter.

␣ Is used to indicate one and only one blank. In most cases, blanks are insignificant and any number of them may be entered between commands and parameters. However, in certain cases, one and only one blank must be entered - this blank is indicated by the "␣" character.

[...] Is used to specify an optional parameter. For example, [<VALUE>] would indicate that VALUE is an optional address parameter. You'll find that many parameters are optional, and in such cases logical default values will be supplied by BUG/65.

or Is used to delimit a list of choices. In such a list, one and only one choice may be used. For example, "+ or -" indicates that you may enter a plus sign or a minus sign, but not both.

filespec Is used to indicate a standard OS/A+ filespec. This consists of the device name followed by a colon and the filename. For example, "D:DATAFILE" is a valid filespec for a file named DATAFILE on disk drive one.

SECTION 3: Address Parameters

BUG/65 allows numerical addresses to be specified in a variety of ways. You can use hexadecimal or decimal notation, add and subtract terms, or add a relocation factor to any address. The following Backus-Naur definitions describe the various address types:

```
<ADDR>    :=   + or - <TERM>  [ + or - <ADDR> ]

<TERM>    :=   <NUMBER>  or  X<NUMBER>

<NUMBER>  :=   <DECNUM>  or  <HEXNUM>

<DECNUM>  :=   .<DECIMAL DIGITS>

<HEXNUM>  :=   <HEXADECIMAL DIGITS>
```

In the above, the only item not literally defined is the "X" item in the definition of a TERM. This is used to indicate that the following NUMBER is to be relocated by adding the value of the current relocation base to the value of NUMBER. The current relocation base is set by the "B" command.

All address parameters are interpreted as 16-bit positive numbers in the range of Ø to 65535. Overflow isn't detected or reported as an error.

Some examples will help (all of these are valid address expressions):

1FA1	a hexadecimal number.
.1ØØ	a decimal number (one hundred).
1ØØØ+.2Ø	a hexadecimal number plus a decimal number. This evaluates to 1Ø14 hex (4116 decimal).
1+2-3+4	a long expression. Evaluates to 4.
X1234	a relocated address. If the current relocation base has the value $1ØØØ, then this expression will evaluate to $2234.

3.1 Spaces as Parameter Delimiters

BUG/65 uses spaces as parameter delimiters. This makes for easier and quicker entry of commands. However, it does introduce some conventions regarding the use of spaces that you must be aware of:

* Spaces may not be embedded in a number. For example, "12 34" is interpreted as two parameters ($12 and $34) and not as the single parameter $1234.

* Spaces aren't allowed between the "X" relocation specifier and it's associated relocated address. For example, "X 1234" is interpreted as two parameters. The first will have the value of the current relocation base and the second is $1234.

* Any number of spaces may be used to separate two parameters. For example, "1234 5678" is a perfectly valid way of entering the two parameters $1234 and $5678.

SECTION 4: Loading and Running BUG/65

BUG/65 is shipped on your master diskette as a relocatable COMmand file, named "BUG65.COM". Therefore, BUG/65 functions just as does any OS/A+ extrinsic command: simply type "BUG65" when OS/A+ prompts with D1: (or Dn: if you have changed default drives...see the OS/A+ manual for more details) and BUG/65 will load into memory and relocate itself to just above the current value of LOMEM (contents of $2E7-$2E8).

4.1 Specifying BUG/65's Load Address

If you need BUG/65 to load at some location other than LOMEM (which is typically around $2000 with OS/A+ version 2 and around $2C00 with version 4), you may also enter a load address on the OS/A+ command line. The address must be in hex, must be at or below $9A00, and should be above LOMEM. Remember, BUG/65 occupies 8K bytes, which means it will occupy memory starting at the address you give and ending $2000 bytes higher.

EXAMPLE:

```
[D1:]BUG65 8000
```

This usage will load BUG/65 at $8000, set its restart point at $8200, and occupy memory from $8000 through $9FFF.

4.2 Creating a Non-Relocatable Version

In order to allow itself to be relocated virtually anywhere in memory, BUG/65 as shipped includes a relocation bit map and a relocation program. In addition, relocatable BUG/65 always loads in at locations $9800 through $BC00. If these addresses are "poison" to you (e.g., if you want to use BUG/65 with a cartridge plugged in), you may wish to produce a non-relocatable version designed to run within an address range you pick.

If so, USING A 48K SYSTEM, simply specify the loadpoint, as shown in the preceding section (e.g, via "BUG65 7000") and allow BUG/65 to load and relocate. Then exit to OS/A+ (via Quit) and use the OS/A+ intrinsic command SAVE to save a non-relocatable version. The address range to be SAVEd may be calculated as follows:

```
SAVE filename.COM loadpoint+$200 loadpoint+$2000
```

Thus, if you had specfied "BUG65 7000", you could save the non-relocatable version via

```
SAVE BUG7000.COM 7200 9000
```

thus also giving it a name which will later remind you where it will load at. To execute this non-relocatable version, simply type in its name (BUG7000 in the example shown).

SECTION 5: Command Entry

When you see BUG/65's input prompt (the ">" character) in the left-hand column of the screen, then you're in command entry mode. Any data typed at that point will be entered into the command line buffer - the command line isn't executed until you type RETURN. You can enter as many commands in one command line as will fit in the command line buffer (100 characters). As soon as you type the RETURN, you'll leave command entry mode and BUG/65 will begin executing the command(s) in the command line.

You can tell the difference between command entry mode and command execution mode. In command entry mode, the cursor is displayed. When a command is executing, the cursor is blanked. If you try to enter more than 100 characters in the command line, BUG/65 will beep the bell and not allow any more characters to be input. At that point, you may either hit RETURN to execute what's in the command line so far, or edit some characters out of the command line with the BACKSPACE key.

5.1 Command Line Editing

When entering commands, you may edit mistakes with the BACKSPACE key. The BACKSPACE will move the cursor one column to the left and delete whatever character was in that column. Unfortunately, the normal system editing facilities aren't supported. This is because of the manner in which BUG/65 does keyboard input.

5.2 Normal and Immediate Type Commands

BUG/65 has two types of commands - normal and immediate. Normal commands are those that don't require interaction with the operator for their execution. Immediate commands do require operator interaction. Normally, you'll never be aware of the distinction between the two types - command entry "flows" without any consideration of the command type required. The only difference is that an immediate command must be the first command entered in a command line. Once an immediate command is entered, BUG/65 will begin interacting with the operator for further input. Since this interaction is required for completion of the command, it doesn't make sense to allow immediate commands to be "stacked" in the middle of a command line for execution between other commands. If you try to enter an immediate command in the middle of a command line, you'll get an "IMMEDIATE ERROR" error message and find yourself back in the command entry mode.

The immediate commands are the "A" command (ASCII memory change), the "S" command (hex memory change), the "X" command (change user registers), and the "Z" command (instant assembler).

5.3 Command Execution

For a normal type command, BUG/65 will begin command execution as soon as you type RETURN. For immediate type commands, BUG/65 will begin command execution as soon as you type the command character (provided that character is the first character in the command line).

5.4 Multiple Commands on a Line

Multiple commands may be entered on the same command line. Normally, successive commands in the command line don't require command separators between them other than at least one space character. The exceptions to this are commands for which an optional parameter is being defaulted. For example, the display memory command ("D") may have an optional parameter specified as the end of the area of memory to be displayed. If that ending parameter isn't specified, BUG/65 will default the end to the start plus eight bytes. If you wanted to enter two successive display commands in the command line without defaulting the end parameters, you could type

```
D 1000 1010 D 2000 2010
```

and no command separators would be required because BUG/65 knows that the "D" command only has two parameters and will interpret further characters in the command line as the beginning of a new command. However, if you wanted to default the ending address of the first display command, then you'd have to insert a command separator so that BUG/65 knows that the first display command is finished. If you didn't do this, then the second display command "D" would be interpreted as the second parameter of the first display command (the end address would be interpreted as $0D. The command separator is a comma, so in this case you would enter the commands as follows:

```
D 1000, D 2000 2010
```

SECTION 6: Command Termination

This section describes the many ways that a command will stop.

6.1 Normal Termination

Once a command line is given to BUG/65 for execution, BUG/65 will execute all of the commands in the line to conclusion before returning to command entry mode. It's possible to instruct BUG/65 to execute a command line "forever" (see Section 8.2), in which case BUG/65 will never come back to command entry mode until you manually intervene (with ESC or BREAK - see Section 6.4)

6.2 Error Termination

If an error occurs in command execution, BUG/65 will beep the bell and display a short error message in English indicating the cause of the error. Command execution will stop and you'll enter the command entry mode. Any commands in the command line after the command which caused the error won't be executed. (You should also be aware that BUG/65 will close any file that has been opened using IOCB number one when any error occurs.) (A complete list of error messages is in Section 14.)

6.3 Command Suspension

Once BUG/65 begins executing a command line, you may temporarily suspend command execution by hitting the space bar. This will put BUG/65 in a "hold" condition, at which point you have two alternatives: you can restart the command by hitting the space bar again, or you can abort the command with ESC or BREAK.

6.4 Command Abort

You can abort any command that is executing (except for the read and write disk commands) by hitting the ESC or BREAK keys. BUG/65 will stop executing the command and you'll enter command entry mode.

6.5 The RESET Key

BUG/65 traps the RESET key so that hitting RESET will bring you back to BUG/65. RESET will stop any command that is executing. You'll see the BUG/65 version and copyright prompt, and you'll be in command entry mode. RESET will reset all of BUG/65's internal stuff except for any user defined or modified parameters. For example, the user's registers, the current relocation base, etc., aren't cleared on a RESET - they'll retain whatever values they had before the RESET. (All of this depends, however, on the fact that the reset vectors haven't been modified by the user - either by using a BUG/65 command or by a user program. If you've modified the reset vectors, then the action of the RESET key is your responsibility.)

6.6 Manual Restart

Since BUG/65 is relocatable, the manual restart point (coldstart) depends upon where it has been relocated to. If you specified an address to load BUG/65 when you gave the OS/A+ command line (e.g., BUG65 4000), then the coldstart point is $200 greater than the address specified, and you may use 'RUN address' from OS/A+ if desired (e.g, RUN 4200 if the original command was BUG65 4000). In any case, you may inspect location $000C (via the BUG/65 command 'D C') to determine the coldstart point. The 6502 word address in locations $0C and $0D (LSB, MSB order) points to BUG/65's restart point. The result of a manual restart is the same as if the default RESET key processing occurred (see section 6.5).

SECTION 7: Command Descriptions

Throughout the descriptions of the commands, comments are sometimes presented in the command line examples. These are denoted by the characters "*/". Anything appearing on a line after these characters is a comment and is NOT part of the command line being exemplified.

The commands are presented in alphabetical order.

7.1 A - Change Memory, ASCII mode

A <ADDR>↵

The A command allows you to replace the contents of memory bytes beginning at location <ADDR> with ASCII characters. As soon as you type the required space character after the address, BUG/65 will prompt you with the current contents of the memory location at <ADDR>. Those contents will be displayed as an ASCII character. At that point, you have the following options:

1. Typing a SPACE will cause the current memory location to be skipped and the contents of the next memory location to be displayed.

2. Typing an UNDERLINE will cause the current address to be decremented by one. The new address is then displayed on the next line of the screen followed by the contents of the new memory location.

3. Typing a RETURN will cause the address of the current memory location to be displayed on the next line of the screen followed by the contents of the current location.

4. Typing ESC will get you out of the command and back into command entry mode.

5. Typing any character other than "@" will cause the ATASCII value of that character to be entered into memory at the current address. The address is then incremented by one and the contents of the new memory location are displayed.

6. Typing the character "@" causes the next character typed to be entered into the current memory location as its pure ATASCII value without any of its control character significance. For example, typing "@ ESC" will insert the ATASCII value for ESC into memory. The address is then incremented by one and operation continues as in 5. above.

After you exercise any option except option 4., BUG/65 will again prompt you with the contents of the current location and you may then choose from any option again.

7.2 B - Set Relocation Base

```
B <ADDR>
```

The B command will set the value of the relocation base to ADDR. The relocation base is intended for use with relocating assemblers. In a relocatable environment, listings typically are addressed from location zero. When a module to be debugged is subsequently loaded into memory, it will have a relocation offset added to the addresses in the listing. The B command allows you to set the relocation base to the load address of the module you're working on and then to reference addresses within the module by simply prefixing each address expression with the relocator symbol "X". For example, suppose that a relocatable module is loaded at location $5380 in memory. Suppose further that we want to display the contents of a memory location which is $230 from the beginning of the module. The following commands would do the job:

```
B 5380, D X230
```

The world isn't overrun with relocating assemblers for the ATARI. However, until it is, the B command has other useful applications. These take advantage of the fact that the relocation base value is a variable which can be modified during command execution. For example, suppose you know that the string of characters "ABCD" is stored somewhere on a diskette and you want to find the sector that contains it. The following commands will do the trick:

```
B 1

D X, R% X 4000 1, L 4000 407F 41 42 43 44, B X+1/
```

This uses some commands not introduced yet, but this is what happens: First, X is set to 1 with one command line. Then a second command line will display memory at the location X (so you'll know where you're at as you step through), read sector number X into memory locations $4000-$407F, locate the string "ABCD" in that sector buffer, then bump X by one for the next sector. The slash at the end of the command line means that the command line will execute forever. What will happen is that BUG/65 will continuously read diskette sectors. For every sector read, you'll see at least a memory display of eight bytes beginning at address X (which is the sector number). If the Locate instruction finds the string "ABCD" in the sector buffer, it will display the location of the string. At that point, just hit ESC to stop the command, and display the value of X ("D X RETURN"). The sector containing the string will either be the value of X or one before it, depending on how fast your ESC was.

7.3 C - Compare Memory Blocks

C <STARTBLOCK1> <ENDBLOCK1> <STARTBLOCK2>

Compare is used to compare the contents of two blocks of memory. The block of memory beginning at STARTBLOCK1 and ending with ENDBLOCK1 is compared to the same size block beginning at STARTBLOCK2. If both blocks are the same, then there will be no output. If any bytes in the blocks differ, then BUG/65 will display a line of data in the following format for every byte that is different:

AAAA = BB CCCC = DD

where AAAA = the hex address of the differing location in the first block, BB = the hex contents of location AAAA, CCCC = the hex address of the differing location in the second block, and DD = the hex contents of location CCCC.

7.4 D - Display Memory

```
D <START> [ <END> ]
```

The D command displays the contents of the memory block beginning at START and ending at END. If END isn't specified, then the default value of START+7 is used. The memory block is displayed in the following format:

```
AAAA = BB BB BB BB BB BB BB BB   CCCCCCCC
```

where AAAA = the hex address of the first byte in this line, BB = the hex contents of successive memory locations beginning at location AAAA, and C = the ASCII character interpretation of the positionally corresponding BB value of the byte.

7.5 E - Execute a Command File

```
E #filespec
```

The E command is used to execute a command line from a command file. The file specified by filespec must consist of a line of BUG/65 commands and parameters and must be ended with an ATASCII EOL character ($9B). BUG/65 will only execute one command line from a command file and then it will stop reading the file. Command files can be chained however, so that the last command in one file can execute another command file. An E command should be the last command in a command line because any commands after the E in the line won't be executed.

7.6 F - Fill a Memory Block with a Value

```
F <START> <END> [ <VALUE> ]
```

The F command will fill the block of memory beginning with START and ending with END with VALUE. If VALUE isn't specified, then zero will be used. Note that VALUE is a byte value - the least significant byte of the 16-bit VALUE will be used for the fill.

7.7 G - Goto a User Program

G [<START>] [@<BRKPOINT> [RN=<VALUE>] [I=<COUNT>]]

The G command will execute a user program beginning at START. If START isn't specified, then execution begins at the current value of the user's PC register. BRKPOINT is an optional breakpoint. If the user's program trys to execute the instruction at BRKPOINT, the program will break back to BUG/65 and BUG/65 will display the contents of the user's registers at that point. Examples:

```
G  1000   /* go at location $1000, no breakpoint

G  @4300  /* go from wherever our PC was and
             break at location $4300 */
```

A breakpoint may be conditionally qualified by a required value in a specified register. "RN=<VALUE>" will tell BUG/65 to break at that point only if the value of user register "N" equals VALUE. If that condition isn't met, then the user's program is allowed to continue executing at the location of the breakpoint. (The instruction that was at the breakpoint location WILL be executed.) The mnemonic names of the registers that may be specified for "N" are: A, X, Y, S, and F, which stand for the user's A, X, Y, Stack, and Status (flags) registers respectively. (Note that only the least significant byte of VALUE is used for this qualification.)

Example:

```
G  1000  @1422  RX=33

   /* go from location $1000 and break at
   location $1422 only if register X
   equals $33 */
```

A breakpoint may also be qualified with an iteration counter. "I=<COUNT>" tells BUG/65 to allow the execution of the instruction at the breakpoint COUNT times before breaking.

Example:

```
G   1000  @2300  I=2

   /* go from location $1000 and break
   the second time we hit the instruction
   at $2300 */
```

The register and iteration qualifications may be used together. In this case, the register condition must be met before the iteration counter is decremented. As in the following example:

```
G  1000  @1234  RA=50  I=3

         /* go from location $1000 and break
         the third time the instruction at loc-
         ation $1234 is executed with register
         A equal to $50 */
```

All of this flexibility isn't without its price, however. Because BUG/65 has to do quite a bit of evaluation at every breakpoint before deciding if the break condition has been met, don't expect to be able to conditionally pass through breakpoint instructions at real-time speed. As long as you never execute the instruction at the breakpoint, you're OK, but as soon as BUG/65 gets the break, expect several hundred instructions to be executed before your program is given back control after the break isn't met.

Also, BUG/65 was NOT designed to allow breakpoints in PROM resident code. If you attempt to set such a break point, or if you try to set a breakpoint at a non-existent memory location, you'll get a "BREAKPOINT ERROR".

One other thing. BUG/65 will automatically remove breakpoints from your program after a break occurs. Breakpoints aren't left set after the break is performed.

7.8 H - Hexadecimal Arithmetic

```
H  <NUMBER1>  <NUMBER2>
```

The H command will calculate the sum NUMBER1 + NUMBER2 and the difference NUMBER1 - NUMBER2 and display the results on the next line of the screen as two hex words. The sum is the first word displayed, the difference is the second.

7.9 I - Display Disk Directory

```
I
```

The I command will display the directory of the diskette in drive one. The display can be suspended or halted with the SPACE or ESCAPE keys respectively.

7.10 J - Create a Command File

```
J #filespec, string
```

The J command allows you to create command files for execution by the E command. The string in the command is any string of valid BUG/65 commands. The string will be written to the file specified by filespec in the format expected by the E command. Please note the comma after the filespec - it's required, else BUG/65 won't know where your filespec stops and your command string starts. Also note that the J command doesn't allow multiple commands in the command line to be executed after the J command - everything in the line after the filespec and up to the RETURN is written to the file instead of being executed.

7.11 K - Convert Hex to Decimal

```
K <NUMBER>
```

The K command will convert NUMBER to a decimal number and display the result on the next line of the screen. NUMBER can be any valid address expression.

To convert decimal to hex, just display memory at the decimal location of the number you want to convert. The hex equivalent of the decimal location appears in the display output as the hex word on the beginning of the line. For example, to convert 1000 decimal to hex, just execute the command "D .1000". You'll see the hex conversion of 1000 as the first hex word on the next line.

7.12 L - Locate a Hex String

```
L <START> <END> <BYTE1> <BYTE2> ... <BYTEn>
```

The L command will search the block of memory beginning at START and ending at END for a hex string. The hex string is defined by BYTE1...BYTEn, which are interpreted as the hex bytes of the pattern string. (Only the least significant bytes of the address values are used for each byte in the string.) Wildcard bytes which will match any byte in memory may be specified by the character "*" in the string. BUG/65 will output the addresses of every occurrence of the string found in the block. For examples:

```
L 1000 10FF 41 42 43

  /* will locate any occur- rences of
  the string "ABC" in the memory block
  $1000 to $10FF */

L 1000 2000 10 * 20

  /* will locate any occur- rences of a
  three-character string which begins
  with $10 and ends with $20 in the
  memory block $1000 to $2000 */
```

7.13 M - Move a Memory Block

```
M <START> <END> <TO>
```

The M command will move the block of memory beginning at START and ending at END to TO. BUG/65 will take care to handle overlapping moves correctly, either for moves up or down.

7.14 P - Select Output Devices

```
P [S] [P]
```

The P command is used to select output to either the screen ("S") or the printer ("P") or to both ("SP").

For example:

```
P S       /* turns screen output on,
             printer output off */

P P       /* turns printer output on,
             screen output off */

P S P     /* turns both screen and
             printer output on

P         /* turns both outputs off -
             commands will still be
             accepted and executed, you
             just won't see their entry or
             output anywhere. */
```

In addition to allowing you to list BUG/65 results to the printer, this command was designed to allow you to debug the generation of intricate screen displays without having the outputs of BUG/65 commands scroll your display off the screen. It is a little crude, and might have a few problems depending on what your program has done to OS, but is handy to have in emergencies. (The LFFLAG and NULFLG bytes in the Configuration Table can help you here - see section 11.)

7.15 Quit to OS/A+ command

```
Q
```

The Q command will coldstart DOS. The results are essentially the same as when you power-up the machine.

7.16 Read Commands

7.16.1 R - Read a File

```
R [ <OFFSET> ] #filespec
```

The R command is used to load binary files. If OFFSET is specified, then OFFSET is added to the load address(es) specified in the file, and the data will be loaded at the loading point(s) plus OFFSET. This allows you to load a file into a different memory location than where it is origined at. After the file is loaded, the load starting point specified in the file is placed into the user's PC register.

BUG/65 supports concatenated binary file sections as described in the DOS 2.0S manual. If such a file is loaded using the OFFSET option, however, ALL file sections will be loaded starting at the load addresses specified in the file plus OFFSET. In addition, the user's PC register will contain the value of the load point of the last file section loaded (not plus OFFSET).

7.16.2 R% - Read Sector(s)

```
R% [ <SECNO> [ <BUFFER> [ <NOSECS> ] ] ]
```

The R% command allows you to read a sector or a group of sectors from a diskette in disk drive number one. SECNO specifies the sector number to be read and defaults to one. BUFFER specifies the buffer the sector is to be read into and defaults to BUG/65's loadpoint plus $2000. NOSECS specifies the number of sectors to read and defaults to one. If more than one sector is specified, then consecutive sectors are read sequentially into memory beginning at BUFFER.

7.17 S - Change Memory, Numeric mode

S <ADDR>␣

The S command allows you to replace the contents of memory bytes beginning at location ADDR with numerical values. As soon as you type the required space character after the address, BUG/65 will prompt you with the current contents of the memory location at ADDR. Those contents will be displayed as a hexadecimal byte value. At that point, you have the following options:

1. Typing SPACE will cause the current memory location to be skipped and the contents of the next memory location to be displayed.

2. Typing an UNDERLINE will cause the current address to be decremented by one. The new address is then displayed on the next line of the screen followed by the contents of the new memory location.

3. Typing a RETURN will cause the address of the current memory location to be displayed on the next line of the screen followed by the contents of the current location.

4. Typing ESC will get you out of the command and put you back into command entry mode.

5. Typing an address value (any valid address expression) will cause that value to be entered into memory at the current address. The address is then incremented by one and the contents of the new memory location are displayed. (Only the least significant byte of the address value will be entered into memory.)

After you exercise any option except option 4., BUG/65 will again prompt you with the contents of the current memory address and you may select any of these options again.

7.18 T - Trace a User Program

```
T [S] [ <COUNT> ]
```

The T command will single-step through user program instructions beginning with the instruction at the current user PC register. The number of instructions to be executed are specified by COUNT, which defaults to one. If "S" is specified, then all of the instructions in a subroutine are counted as one instruction for tracing purposes - the trace is turned off until return from the subroutine ("S" stands for "skip the subroutine"). After every instruction traced, BUG/65 will display the contents of the user's registers.

Some examples:

```
T       /* will execute one instruction and then
           display the register contents */

T 5     /* will execute five instructions,
           displaying registers after each
           instruction */

TS 10   /* will execute 16 instructions. If any
           of the instructions are JSR's, then the
           trace will be turned off after the JSR
           until the subroutine executes an RTS */
```

The trace command can't be use to trace instruction execution through PROM resident code. Any attempt to do so, or to trace through non-existent memory, will result in a "BREAKPOINT ERROR".

7.19 U - Call a User Subroutine

```
U <ADDR> [ <PARAM> ]
```

The U command is used to call a user subroutine at ADDR. The user routine is passed the optional parameter PARAM in the X register (low byte) and Y register (high byte). The user routine should return to BUG/65 via an RTS instruction. If PARAM isn't specified, then zero is used.

7.20 V - Display User's Registers

```
V
```

The V command will display the contents of the user's registers in the following format:

```
 A  X  Y SP NV_BDIZC   PC  INSTR
HH HH HH HH BBBBBBBB HHHH  LDA  1000,X
```

This is interpreted as follows:

```
A  =  the hex value of the A reg
X  =  the hex value of the X reg
Y  =  the hex value of the Y reg
SP =  the hex value of the stackpointer
N  =  the binary value of the negative flag
V  =  the binary value of the overflow flag
   =  the binary value of an unused bit in the
B  =  the binary value of the break flag
D  =  the binary value of the decimal flag
I  =  the binary value of the interrupt enable bit
Z  =  the binary value of the zero flag
C  =  the binary value of the carry flag
PC =  the hex value of the PC reg (This is a
      pseudo register maintained by BUG/65.
      It contains the location of the next
      user program instruction to be executed.)
INSTR = the instruction at the current PC
```

7.21 Write Commands

7.21.1 W - Write a File

```
W [ :A ] <START> <END> #filespec
```

The W command is used to write a binary file. Memory from START to END is written to the file specified by filespec in the standard OS/A+ binary file format. If the ":A" option isn't specified, then the data written will replace the current contents of the file if the file already exists. If the ":A" option is specified, then the data is appended to any data already in the file. A load header consisting of a start and end address as described in the OS/A+ manual will precede the appended data.

7.21.2 W% - Write Sector(s)

```
W% [ <SECNO> [ <BUFFER> [ <NOSECS> ] ] ]
```

The W% command is used to write a sector or a group of sectors to a diskette. SECNO specifies the sector number to be written and defaults to one. BUFFER specifies the memory location of the sector data to be written and defaults to the BUG/65 loadpoint plus $2000. NOSECS specifies the number of sectors to be written and defaults to one. If more than one sector is specified, then consecutive sectors are written sequentially from memory beginning at BUFFER.

7.22 X - Change User's Registers

```
X REGNAME
```

The X command allows you to change the contents of user registers. REGNAME is a one-character register name mnemonic. The allowed register names and their meanings are:

```
A = A register
X = X register
Y = Y register
S = stackpointer register
P = program counter pseudo-register
F = status register (flags)
```

After you type in the name of the register to be changed, BUG/65 will prompt you with that name character followed by an equals sign. At that point you have the following options:

1. Enter the new value for the register. The new value may be any valid address expression. After the new value, typing RETURN will end the command. Or you can type SPACE which will prompt you with another register name for possible change. The next register name is determined by the order of the above list. For example, if you change register Y then hit a space after the new value, BUG/65 will prompt you for possible change of register S. This prompt list continues through register F and then wraps back to register A again.

2. Enter RETURN or ESC to end the command. BUG/65 will display the new contents of the registers and then put you back into command mode.

7.23 Y - Disassemble Memory Block

Y <START> <END>

The Y command will disassemble instructions in memory beginning at START and ending at END. The following conventions are used in the disassembly:

1. Standard MOS Technology mnemonics are used for opcodes.

2. Illegal opcodes are displayed as "***".

3. All numeric operands are displayed as hexadecimal numbers.

4. Zero page operands will display as two hex digits, all other non-immediate operands will display as four hex digits.

5. No operand is displayed for accumulator mode operands.

7.24 Z - Instant Assembler

Z <ADDR>↵

The Z command allows you to assemble instructions to be stored in memory at ADDR. Immediately after typing the SPACE character (or RETURN, which is allowed as well), BUG/65 will prompt you with the current program counter value of the instant assembler (which initially will be ADDR). At that point you may type in a valid assembly language instruction. The format for an instruction line is:

[<LABEL>] <OPCODE> [<OPERAND>]

LABEL may be any label in the form "Ln", where "n" may be any digit from zero to nine. OPCODE may be any valid MOS Technology instruction mnemonic or one of two pseudo-ops (described below). OPERAND, if allowed by the addressing mode of the instruction, may be any valid address expression. At least one space must separate a label from an opcode or an opcode from an operand.

After typing your instruction, type RETURN and the instruction will be entered into memory at the current PC if it doesn't contain any errors. If there are any errors, then BUG/65 will display an error message and will reprompt you with the current (unchanged) PC. If there are no errors, then BUG/65 will display the object code created by the instruction to the right of the instruction on the screen and will prompt you with the PC of the next instruction on the next screen line. You may exit the instant assembler by typing ESC at any time, or by typing RETURN by itself in response to the PC address prompt.

The instant assembler provides you with two pseudo-ops. "/" followed by an address will change the PC to that address. It acts like an ORG ("*=") pseudo-op. For example, "/4000" will set the PC of the next instruction location to $4000. "+" followed by an address will insert the value of that address (least significant byte) at the current PC and bump the PC by one. It acts like a DB (.BYTE) pseudo-op. For example, "+34" will insert the hex byte 34 at the current PC.

The instant assembler provides a simple labeling capability. You may prefix an instruction with a two character label of the form "Ln", where "n" may be any digit from Ø-9. You may then use that label as an operand in an instruction, with the following three restrictions:

1. Immediate type operands (#HH) can't be labels.

2. Indirect type operands can't be labels.

3. A label can't be combined with any of the standard address operators (+, -, X, etc.)

Label references may be forward or backward. BUG/65 will store unresolved references and resolve them when the label is later defined. You may reference undefined labels twenty times before BUG/65 runs out of room to store the unresolved locations - you'll then get an error message and the assembly will be aborted. The same label may be reused more than once. In such cases, BUG/65 will use the last defined address of the label when it is referenced.

If any labels have been referenced but not defined when you exit the instant assembler, BUG/65 will prompt you with a message and the label name followed by an equals sign. At that point you may either define the label by entering any valid address expression followed by a RETURN, or you may chose not to define it and simply hit RETURN. If you don't define the label, then the value of the label is defaulted according to the following two rules.

1. If an instruction using the undefined label is a relative branch, then the value of the label for that instruction defaults to the location of the instruction plus two.

2. For all other instructions, the value of the label defaults to the location of the instruction plus three.

These rules guarantee that all branching instructions using undefined labels are effectively turned into NOP'S. This offers some measure of protection against a program going into never-never land. (If you reference a label that isn't yet defined, the object code displayed to the right of the instruction on the screen will show addresses generated according to these rules. Don't worry, when the label is subsequently defined, BUG/65 goes back and fixes up all these references.)

SECTION 8: Special Command Modifiers

8.1 Repeat Last Command Line

{RETURN}

The last command line entered and executed may be repeated without typing the whole thing in again - just hit RETURN. BUG/65 remembers the last line entered for just this purpose.

8.2 Repeat Command Line Forever

/

Appending a slash to the end of a command line will cause BUG/65 to repeat the execution of that command line forever. The only way to stop such a repeat is to suspend or abort the command.

8.3 Display Last Command Line

=

If you want to see what your last command line was, possibly because you might want to repeat it, just type the "=" character as the first character of the new command line. BUG/65 will display the last line entered for you.

SECTION 9: BUG/65 Memory Protection

BUG/65 won't allow you to modify any portion of it's code or variable storage areas with a BUG/65 command. Any attempt to do so will result in a "PROTECTION ERROR". For example, if we assume that the BUG/65 was loaded via the command "BUG65 2000", the following command will cause an error because it attempts to move a memory block into BUG/65's area:

```
M  4000  40FF  2000
```

BUG/65 protects all memory from loadpoint to loadpoint+$1FFF in this manner, where loadpoint is that specified in the invoking OS/A+ command line (or LOMEM, if no loadpoint is specified). (The memory protection feature can be turned off by changing a byte in the Configuration Table.)

SECTION 10: BUG/65 Memory Usage

BUG/65 uses memory from $80 to $XX and loadpoint to loadpoint+$01FF for variable storage. You can determine the value of XX by looking at the LSTPG0 byte in the Configuration Table. It uses memory from loadpoint+$200 to loadpoint+$1FFF for code storage.

10.1 Page Zero Sharing

BUG/65 will share the page zero memory that it needs with a user program. It does this by keeping two copies of these page zero locations. When BUG/65 is running, the BUG/65 page zero locations contain BUG/65's stuff. When a Go is done to a user program, BUG/65 will save it's own page zero data and replace it with the user's data. If a user program breaks back to BUG/65, the reverse operation is performed.

In addition, BUG/65 will translate any command reference to these shared page zero locations so that the user may modify or inspect his own page zero data. It does this by translating any command reference to the user's page zero data to the location where the user's copy of the data is actually being stored. This is all transparent to the user. For example, you can fill memory from $80 to $FF with zeros without crashing BUG/65. If you then display $80 to $FF, you will see zeros. They aren't really in locations $80 to $FF of course, but they will be when you run your program. (This is the reason it may seem to take an extraordinarily long time to perform certain commands (Fills, for example). The reason is that every memory reference has to go through this translation process - both to translate zero page references if necessary and to check to make sure that BUG/65 isn't being overwritten.)

SECTION 11: Customization with the Configuration Table

There is a Configuration Table located near the beginning of the code segment of BUG/65. By changing this data, you can customize some BUG/65 stuff. In the table which follows, "+$xxx" means that the configuration value is located $xxx bytes above the loadpoint address, where loadpoint is the address specified in the invoking OS/A+ command line (or LOMEM, if loadpoint is not specified). Example: if the invoking command was "BUG65 6000", then DISPV will be located at $6209.

NAME	LOCATION	FUNCTION/COMMENTS
DISPV	+$209	A JMP instruction to BUG/65's display a character routine. All chars displayed on the screen go through here. The char to be displayed is passed in reg A.
PRINTV	+$20C	A JMP instruction to BUG/65's print a character routine. All chars sent to the printer go through here. The char to be printed is passed in reg A.
GETKYV	+$20F	A JMP instruction to BUG/65's get a keyboard character routine. All keyboard reads go through here. The key read is returned in reg A.
TSTKYV	+$212	A JMP instruction to BUG/65's test for a key waiting routine. All tests for key waiting go through here. If no key is waiting, the equal flag is returned set. (The key is NOT returned by this routine - GETKYV will be called to read the key if there's one waiting.)
BEEPV	+$215	A JMP instruction to BUG/65's bell routine. All beeps are generated through here. To eliminate the beeps, just patch this out with an RTS.
CHRCLR	+$218	Character background color byte value.
CHRLUM	+$219	Character luminance byte value.
BRDCLR	+$21A	Border color byte value.
EOLBYT	+$21B	This is the byte sent to the printer at the end of a line. Normally set to 0DH or 9BH.

LFFLAG	+$21C	If nonzero, then a linefeed character is sent to the printer after every EOLBYT.
NULFLG	+$21D	If nonzero, then 40 nulls will be sent to the printer after every line. Used to flush the printer buffer maintained by the ATARI OS so that all lines will print immediately.
PROTFG	+$21E	If nonzero, then BUG/65 will not allow itself to be overwritten with a BUG/65 command. If zero, then BUG/65 will allow itself to be modified.
MCBEND	+$21F	High byte of end address of BUG/65's code. Normally set to high byte address of loadpoint+$2000 (e.g, $50 if the invoking OS/A+ command were BUG65 3000). You would change this if you added any user command handlers after BUG/65. The handlers would then be included in BUG/65's memory protection features.

To change anything in the Configuration Table, you must first disable memory protection by writing a small program to stuff a zero into PROTFG. For example, assuming that the loadpoint is $2000 (command line was BUG65 2000), then using the instant assembler, you could enter "LDA #0, STA 221E, RTS" at location $5000, and then run the program with the "U" command by entering "U5000 <RETURN>". This will disable memory protection. Then make your changes, reenable memory protection if you want by storing $FF into PROTFG, then dump the modified BUG/65 to diskette.

Be careful when changing any of the JMP instruction vectors. Since BUG/65 is constantly calling these locations, the instant you change them control will be passed to the new routine. Your replacement routines had better be in place and ready to run or it's ga-ga time. Actually, you will probably have to change all three bytes of a vector at once with a small user program.

Also, be careful about calling the vectors DISPV, PRINTV, GETKYV, TSTKYV, and BEEPV. Since they use BUG/65's page zero data to operate, they can't be called from a running user program without first calling the MCBGP0 routine defined in the User Program Interface section.

SECTION 12: User Command Interface

It's possible to add commands to BUG/65. The hooks to do so have been provided in a group of vectors located at loadpoint+$0220 called the User Command Interface Vectors. These vectors provide most of the interfaces to BUG/65 that you'll need to add commands.

The commands you add may be activated by any non-BUG/65 command char. For example, you could add the numeric commands "1" through "9". When BUG/65 recognizes a non- alphabetic command character, it will call the vector USERCMD. In it's initial state, USRCMD is just a 3-byte subroutine that returns the equal flag reset. BUG/65 assumes that the equal flag being reset means that a user command handler considers the command illegal. In this case, BUG/65 will report a "CMD ERROR". If USRCMD returns the equal flag set, then BUG/65 assumes that a user command handler processed the command. In this case, BUG/65 won't generate a command error, and will proceed to process the rest of the command line.

So, to add your own command handler, just patch a JMP to your handler at USRCMD. BUG/65 will pass you the command character that it considered illegal in reg A. On return, you must indicate the status of the command - equal set means you handled it, equal reset means you didn't like it either.

There are a number of other vectors in the User Interface group which you may use to process the command. Here's the complete list (and, as in the previous section, the string "+$xxx" indicates a displacement from the loadpoint):

NAME	LOCATION	FUNCTION/COMMENTS
USRCMD	+$22Ø	Subroutine called by BUG/65 on every non alpha comand char. Returns equal set if command handled by user, else equal reset.
GETCHR	+$223	User handler can tell this to get the next char from the command line in reg A.
PUTCHR	+$226	User handler can call this to return the last char taken from the command line. The char itself doesn't have to be passed. This is used to put chars back that you've taken but don't want - like an EOL.
GET1HX	+$229	User handler can call this to collect a hex address from the command line. The address is returned in a word at $FE,$FF. If next command line chars are not a valid address, zero is returned.
GET2HX	+$22C	User handler can call this to collect two hex addresses from the command line. The first address is returned in a word at $FC,$FD, the second at $FE,$FF. Zero is returned for any invalid address.
GET3HX	+$22F	User handler can call this to collect three hex addresses from the command line. The first address is returned in a word at $FA,$FB, the second at $FC,$FD, and the third at $FE,$FF. Zero is returned for any invalid address.

ADRCHK	+$232	User handler can call this to perform the usual BUG/65 address checking and translation. The checking refers to not allowing BUG/65 to be overwritten. The translation refers to correcting user page zero addresses. The user handler passes the address to check in reg X (LO) and reg Y (HI). If the address points into BUG/65, a "PROT ERROR" will occur, and the user handler will not be returned to. If the address references a user page zero value that is being stored somewhere else by BUG/65, then the address of where the actual user page zero byte is located will be returned in reg X (LO) and reg Y (HI).
ERRPAR	+$235	The user handler can JMP to here to report a parameter error. There is no return back to the user handler. BUG/65 will abort command line processing.
DHXBYT	+$238	The user handler can call this to display a hex byte. The byte is passed in reg A.
DHXWRD	+$23B	The user handler can call this to display a hex word. The hex word is passed in reg X (LO) and reg Y (HI).
CTBPTR	+$23E	This is a pointer to BUG/65's jump table for the alphabetic comands. Every letter has a word entry in this table. The entry is the address of the handler for that command minus one. The first word in the table is the address minus one for the "A" command, the last is the same for the "Z" command. If you want, you can change this table to point to your own comand routines, thereby changing the BUG/65 command set.
LSTPGØ	+$24Ø	This is the address (byte value) of the last page zero location used by BUG/65. You can use this to locate free page zero memory for your own use. (See the example user command listing.).

**** SPECIAL NOTE ****

All of the above routines assume that BUG/65 data is in page zero. THEY WILL NOT WORK if called from a running user program for that reason, unless the user program manages page zero with the following two routines:

MCBGPØ	+$241	Assumes BUG/65 data is in page zero. Saves BUG/65 page zero and replaces with user page zero. Use this routine from a running user program before calling any of the above routines.
USERPØ	+$244	Assumes user data is in page zero. Saves user page zero and restores BUG/65 page zero. Use this routine from a running user program after calling any of the above routines to restore the running program's page zero data.

12.1 User Command Handler Example

Here is an assembly listing of an example user comand. This command will be command "1". It will calculate and display an exclusive-or checksum byte on a range of memory. The syntax of the command is:

```
1 <START> <END>
```

NOTE: It is highly recommended that user commands only be patched into a non-relocatable version of BUG/65. See Section 4.2 for instructions on making a non-relocatable version with a user specified loadpoint.

```
;*************************************************
;  EQUATES INTO BUG/65:

loadpoint =  ????          to be determined by user!!
lp = loadpoint            just an abbreviation
MCBEND  =    lp+$21F       BUG/65 END CODE MSB
DISPV   =    lp+$209       DISPLAY CHAR
USRCMD  =    lp+$220       USER COMMAND VECTOR
GET2HX  =    lp+$22C       GET 2 HEX PARAMS
HEX1    =    $FC           HEX PARAM 1 RESULT
HEX2    =    $FE           HEX PARAM 2 RESULT
ERRPAR  =    lp+$235       REPORT PARAM ERROR
DHXBYT  =    lp+$238       DISPLAY HEX BYTE
LSTPGØ  =    lp+$240       LAST BUG/65 PØ BYTE USED
EOL     =    $9B           END OF LINE CHAR
```

```
;*************************************************
            *=    USRCMD          PATCH US INTO BUG/65
            JMP   USERC1
;
            *=    1p+$2000        RIGHT AFTER BUG/65 CODE
USERC1      CMP   #'1             COMMAND "1" ?
            BEQ   CMDOK           YES
            RTS                   ELSE RTN EQUAL RESET - ERR
;
CMDOK       JSR   GET2HX          GET START, END
            LDA   HEX1            MAKE SURE BOTH SPECIFIED
            ORA   HEX1+1
            BEQ   PARMER          OR ELSE ERROR
            LDA   HEX2
            ORA   HEX2+1
            BNE   PARMOK
;
PARMER      JMP   ERRPAR          REPORT PARAM ERROR
;
PARMOK      LDX   LSTPGØ          LAST BUG/65 PØ BYTE
;                                   (WE'LL USE THE NEXT
;                                   FOR OUR ACCUMULATOR)
            LDA   #Ø               CLEAR ACCUMULATOR
            STA   1,X
            TAY                    INIT Y PTR INDEX
;
LOOP        LDA   HEX2+1           PAST END ADDRESS ?
            CMP   HEX1+1
            BCC   DONE             YES
            BNE   NXTEOR           NO
            LDA   HEX2
            CMP   HEX1
            BCC   DONE             YES
;
NXTEOR      LDA   (HEX1),Y         CALC EOR CHKSUM
            EOR   1,X              EOR WITH ACCUM
            STA   1,X              AND SAVE IN ACCUM
            INC   HEX1             BUMP PTR
            BNE   LOOP
            INC   HEX1+1
            JMP   LOOP
;
DONE        LDA   #EOL            TO NEXT SCREEN LINE
            JSR   DISPV
            LDX   LSTPGØ          RESTORE ACCUM ADDRESS
            LDA   1,X             DISPLAY HEX RESULT
            JSR   DHXBYT
            LDA   #Ø              RTN OK (EQUAL SET)
            RTS
;
            *=     MCBEND          CHANGE BUG/65 CODE
            .BYTE  >[*+$FF]        END BYTE TO INCLUDE
            .END                  THAT'S ALL FOLKS
```

SECTION 13: Error Messages

The following is a list of all of the error messages and a short explanation of each one:

COMMAND ERROR
: An attempt to execute an illegal command. A letter or number that isn't a valid command mnemonic was interpreted as a command character. For example, trying to execute the command "N" will cause a command error.

IMMEDIATE CMD ERROR
: An attempt to execute an immediate type command in the middle of a command line. An immediate command (A, S, X, or Z) must be the first command on a command line. See section 5.2.

PROTECTION ERROR
: An attempt was made to modify BUG/65's code or variable memory areas with a BUG/65 command.

PARAM ERROR
: Caused by the usage of any invalid command parameter.

REGISTER ERROR
: An invalid register name was specified in either the G or X command.

BREAKPOINT ERROR
: An attempt was made to set a breakpoint in either PROM memory space or non-existent memory.

PRINTER ERROR
: Any printer error returned to BUG/65 by the operating system. (BUG/65 uses the ATARI OS to print characters. Any error returned by the OS on a print character call will cause this error.)

SYNTAX ERROR
: Caused by an error in the syntax of a command.

I/O ERROR - NNN
: Any disk I/O error returned to BUG/65 by the operating system. (BUG/65 uses the OS/A+ to do disk I/O. Any error returned by the OS/A+ call will cause this error.) NNN is the decimal error number returned by the OS. Refer to your OS/A+ manual for the meanings of these numbers.

*** ERROR - MNEMONIC
In the instant assembler, an invalid opcode mnemonic was entered.

*** ERROR - OPERAND
In the instant assembler, an invalid instruction operand was entered.

*** ERROR - RANGE
In the instant assembler, a branch out of range was attempted.

*** ERROR - TOO MANY LABEL REFS
In the instant assembler, too many references have been made to an undefined label. BUG/65 2.Ø allows twenty references to undefined labels before it's label buffer overflows.

*** ERROR - UNDEFINED - Ln
In the instant assembler, a label has been referenced but not defined. "n" is the label number that needs definition.

APPENDIX

This section applies only to those users who own version 4 of OS/A+.

The version of BUG/65 which you received is not directly compatible with version 4 of OS/A+. Included on your disk, however, is a program which converts the BUG65.COM file into a form which will work under version 4. This program, BUGV4FIX.COM, is a binary program that modifies the relocatable version of BUG65.

The resultant version of BUG65.COM will work ONLY with version 4. Further, under version 4, the R (read binary file) command will not work properly under all conditions. We suggest instead using the OS/A+ LOAD command for loading binary files into memory, although the ERROR 136 produced by the R command may simply be ignored, if desired. Only location $ØØ is improperly affected by this error.

HOW TO USE THE PROGRAM:

1) Copy the files BUG65.COM and BUGV4FIX.COM to a version 4 disk using the COPY24 command (see the OS/A+ manual for details on this command).

2) At the version 4 "D1:" prompt, type the command:

   BUGV4FIX [RETURN]

3) The file BUG65.COM on that disk is now compatible with version 4 of OS/A+.

WARNING: Do NOT perform the BUGV4FIX command on your version 2 master disk!