ATARI DOS FOUR
TECHNICAL REFERENCE MANUAL

CONTENTS

APPLICATION MEMORY LOCATIONS

This chapter describes the fixed memory locations within DOS 4 that are indended to be used by application programs. These memory locations are guaranteed not to change in future versions of DOS 4.

DISK DRIVE NUMBER INDIRECTION

DOS 4 has the ability to support up to eight physical drives and up to ten logical drives. The physical drives are the actual pieces of hardware, and they are numbered from 1 to 8. The logical drives are the drives which are referred to in file specifications, and they are numbered from D0: to D9:. Whenever you give a file specification, DOS 4 must read the logical drive number and decide which physical drive you are referring to. Normally, D1: through D8: refer to physical drives 1 through 8 respectively, while D0: and D9: are not supported. However, by modifying certain memory locations within DOS 4, you may make each logical drive refer to whichever physical drive you wish.

Drive number indirection is controlled by DTYPE at [$73F,$A]. The four low-order bits of memory location DTYPE+n gives the physical drive number which is to be associated with the logical drive number Dn:. If the four low-order bits of DTYPE+n are zero then logical drive Dn: will not be supported.

The contents of DTYPE may be changed at any time, even if there are open files to the drives in question. The physical drive associated with a file is determined when the OPEN statement is executed, so that changing DTYPE will not cause an open file to start reading from a different drive. When changing DTYPE, use AND and OR operations to change the four low-order bits without changing the four high-order bits.

BUFFER ALLOCATION

Buffer allocation is controlled by the contents of memory locations BUFMAX at [$710,1] and BUFSIZ at [$711,1]. BUFMAX contains the number of buffers which are to be allocated; it must be between 2 and 16 (decimal) inclusive. If you want to have N open files on a system with D disk drives then the minimum number of buffers required is N+MIN(N,D). The standard number of buffers is 5.

BUFSIZ determines the size of the buffers. It must contain either 0 (for 256-byte buffers) or

$80 (for 128-byte buffers). If you have any double-density drives (or single-density drives with a two-sector VTOC) then BUFSIZ must contain 0.

The standard value of BUFSIZ is 0. It is recommended that you use a value of 0 even if you only have single-density drives.

The contents of BUFMAX and BUFSIZ may be changed only when there are no open disk files. The recommended procedure is to (a) make sure all disk files are closed, (b) store new values into BUFMAX and/or BUFSIZ, (c) load the Disk Utility Package, (d) use the WRITE DOS command to write out the new version of DOS 4, and (e) re-boot the system.

THE RESIDENT BINARY LOADER

The resident File Management System contains a program which can load and run a machine-language program in the standard binary load file format. The loader has two entry points: LOADER at $70A and KERNEL at $70D.

Entry point LOADER should be used if (a) you know that the loaded program is not going to overwrite the calling program, or (b) you know that the loaded program is not going to return, or (c) you are chaining programs and the loaded program is the next one in the chain.

When you are not chaining programs, the calling sequence for LOADER is:

```
LDY #FILE&255 ;FILE contains the filespec
LDA #FILE/256
LDX #$FF ;Use #$FF to load and run,
         ;#0 to load and not run
JSR LOADER ;Call the loader
CPY #0 ;Error status returned in Y
BMI ERROR ;Branch if error
```

If you are chaining programs, the calling sequence for LOADER is:

```
PLA
PLA ;Cancel subroutine call
LDY #FILE&255 ;FILE contains the filespec
LDA #FILE/256
LDX #$FF ;Use #$FF to load and run,
```

```
                    ;#0 to load and not run
            JMP LOADER ;Go to the loader
```

Entry point KERNEL is used in the case that the loaded program may overwrite the calling program and then return. If this should happen, KERNEL will automatically load and run the Command Processor when the loaded program returns.

The calling sequence for KERNEL is the same as for LOADER, except that before calling KERNEL you must store a value into DUPFLG at [$736,1] and optionally into DUPLO at [$732,2] and DUPHI at [$734,2].

Storing zero into DUPFLG will force KERNEL to load the Command Processor when the loaded program returns.

Storing any non-zero value into DUPFLG will make KERNEL load the Command Processor only if the calling program was overwritten during the load. In this case the calling program should store the address of its first byte into DUPLO and the address of its last byte into DUPHI. If the calling program is not overwritten during the load then KERNEL will return to the calling program instead of loading the CP.

If KERNEL decides that it must load and run the Command Processor then, before it does so, it will store the error status resulting from the load into BLDFLG at [$737,1]. The CP will examine BLDFLG and issue an error message if the contents of BLDFLG indicates that an error occurred during the binary load process.

Note that both LOADER and KERNEL use IOCB #1 to perform the load. Therefore, before calling either of these routines, you must make sure that IOCB #1 is closed.

THE STANDARD BINARY LOAD FILE FORMAT

DOS 4 uses the same format for binary load files as Atari DOS 2.0. A binary load file consists of one or more "segments", each of which gives the data to be loaded into a contiguous block of memory. A segment consists of three parts:

1. A two-byte file type code, in which each byte contains $FF. This is required on the first segment in the file, but optional on subsequent segments.

2. A four-byte header in which the first two bytes give the address where the first byte of the data block is to go and the last two bytes give the address where the last byte of the data block goes.

3. A data block which contains one or more bytes of data that are to be loaded into memory.

There are two memory locations which have special significance to the loader: INIVEC at [$2E2,2] and RUNVEC at [$2E0,2]. Every time the loader finishes loading a segment, it checks to see if a non-zero address was loaded into INIVEC. If so, then the loader immediately executes a subroutine call (JSR statement) to the address in INIVEC. When the program returns, the loader will continue the load.

Note that at the time the loader executes a subroutine call to the address in INIVEC, IOCB #1 will be open for input from the file being loaded. This has two consequences: (1) The program should not attempt to use IOCB #1. (2) If the program wishes to chain to another program, it must close IOCB #1 before calling the loader.

When the entire file has been loaded, the loader checks to see if a non-zero address is in RUNVEC. If so, the loader executes a subroutine call (JSR statement) to that address. The loader will close IOCB #1 before calling the address in RUNVEC.

COMMAND PROCESSOR FILENAME

DUPSPC at [$722,$10] contains the name of the file which contains the Command Processor, normally D1:QDUP.SYS.

You may change DUPSPC at any time. The filename must end with a carriage return ($9B). Use the WRITE DOS command in the Disk Utility Package to write out the new version of DOS 4.

If you wish to replace the Command Processor with a program of your own, all you have to do is to place the name of the file containing your program into DUPSPC. One possible use for this capability is to replace the Command Processor with a batch processor.

Before it loads the CP, the resident FMS will close all IOCB's. The FMS does not clear the screen, so if a screen clear is desired then the CP must do it. If the CP is going to use the resident screen editor "E:" then it should begin with an initialization routine which (a) sets the screen margins, (b) opens IOCB #0 to E:, and then (c) delays long enough to allow vertical blank to bring up the screen.

There are two entry points which can cause loading of the CP: indirectly through DOSVEC at $0A, or directly through KERNEL at $70D. The CP can determine which entry point was used by examining BLDFLG at [$737,1]. The contents of BLDFLG will be zero if entry DOSVEC was used, and non-zero if entry KERNEL was used. In the latter case, the contents of BLDFLG is the error status code resulting from the binary load process; a value greater than or equal to $80 indicates that an

error occured.

Memory location DUPRES at [$738,2] and location [$700,1] (no name assigned) are reserved for use by the Command Processor or its replacement. The standard Command Processor does not use these locations. These memory locations are guaranteed to be zero at power-up and are not subsequently modified by any other program (except that the Disk Utility Package sets DUPRES to zero at the start of the WRITE DOS command and then restores the value of DUPRES at the end of the command).

SIO/PIO COMMANDS AND INTERCEPTION

The following locations contain the serial or parallel bus commands used by DOS 4 to communicate with the disk drive:

symbol	location	command	standard value	alternate value
WRCOMD	[$73A,1]	data write	$50	$57
RDCOMD	[$73B,1]	read	$52	-
DWCOMD	[$73C,1]	directory write	$57	$50
STCOMD	[$73D,1]	status	$53	-

WRCOMD is the command used to write data sectors, and DWCOMD is the command used to write directory and VTOC sectors. Use $50 for fast write (write without verify) and $57 for slow write (write with verify).

Memory location RRVECT at [$7D1,3] contains a JMP instruction. Immediately before each call to SIO or PIO, the FMS executes a subroutine call (JSR statement) to RRVECT. This call takes place after the Device Control Block has been set up. By storing an address into RRVECT+1 and RRVECT+2, you can effectively intercept all calls to SIO or PIO.

The intended use of RRVECT is to allow programs to remain responsive to the user during long disk operations. A typical RRVECT subroutine might check to see if the user has pressed a key and, if so, echo the key to the screen. An RRVECT routine cannot call CIO, but it can make direct calls to the resident screen handlers.

If for some reason an RRVECT routine wants to prevent the FMS from calling SIO or PIO, it can do so by popping the return address off the stack, adding 3 to it, and then pushing it back on the stack. If this is done, then the RRVECT routine must return with an error code in the 6502 Y-register and in the OS variable DSTATS.

The contents of RRVECT+1 and RRVECT+2 is initialized to point to an RTS during both coldstart and warmstart.

ENVIRONMENT CONTROL

CTBOOT at [$7D5,1] indicates whether the computer should enter the cartridge environment or the disk environment at coldstart. If CTBOOT contains zero then the Command Processor will assume control when the system is booted (even if there is a cartridge installed). If CTBOOT contains a nonzero value then the cartridge will assume control when the system is booted, provided that there is a cartridge installed (if there is no cartridge installed then the Command Processor will assume control). The default value of CTBOOT is $FF.

CRTENV at [$7D4,1] indicates whether the computer should enter the cartridge environment or the disk environment at warmstart. If CRTENV contains zero then the Command Processor will assume control when SYSTEM RESET is pressed (even if there is a cartridge installed). If CRTENV contains a nonzero value then the cartridge will assume control when SYSTEM RESET is pressed, provided that there is a cartridge installed (if there is no cartridge installed then the Command Processor will assume control).

The FMS initializes CRTENV during coldstart by copying the value of CTBOOT into CRTENV. The FMS stores 0 into CRTENV every time it loads the CP. The CP and DUP store $FF into CRTENV whenever the RUN CARTRIDGE command is executed. The machine-language programs generated by the GOBASIC utility also store $FF into CRTENV.

Any program loaded from the Command Processor which passes control to a cartridge should store a nonzero value into CRTENV immediately before it passes control.

RELAXATION OF FILESPEC RULES

REQEOL at [$7D6,1] determines whether or not DOS 4 strictly enforces the rules for filespecs. When the value of REQEOL is between $80 and $FF (inclusive), filespec rules are strictly enforced. When the value of REQEOL is between 0 and $7F (inclusive), filespec rules are relaxed. When filespec rules are relaxed, any character that could not be part of a valid filespec is interpreted to mean end-of-filespec.

The default value of REQEOL is $FF. The power-on disk configuration program CONFIG.SYS examines REQEOL and, if REQEOL is non-zero, stores $FF into REQEOL. Thus, values of $01 through $7F can be used to "temporarily" relax the rules for filespecs.

DOS 4 MEMORY MAP

$0A-	$0B	CP Load And Run Vector
$0C-	$0D	FMS Init Vector
$1A-	$1B	DUP Zero Page
$43-	$46	FMS Zero Page
$2E0-	$2E1	Binary Load Run Vector
$2E2-	$2E3	Binary Load Init Vector
$701-	$17FB	File Management System
$17FC-		Sector Buffers
$1BFC-	$20FB	Command Processor
$20FC-	$4FFB	Disk Utility Package

Note 1: DOS 4 assumes that the FMS Zero Page is not altered in between calls to the FMS. Atari DOS 2.0 does not make this assumption. Therefore, a program which uses memory locations $43-$46 may work with Atari DOS 2.0 but not with DOS 4.

Note 2: The actual top address of the DUP varies depending on the Disk Configuration Files that have been merged into it. However, the top address will never exceed $4FFB.

DOS 4 DISK FILE STRUCTURE

This chapter describes the way in which DOS 4 disk files are organized on the disk.

Data on the disk is physically organized into "sectors". Depending on the disk drive being used, sectors contain either 128 bytes apiece or 256 bytes apiece. DOS 4 considers all disks with 128-byte sectors to be "single-density", and all disks with 256-byte sectors to be "double-density", regardless of the recording technique which is acually used by the hardware.

DOS 4 logically organizes its data into "blocks". Each block consists of an integral number of sectors. The number of sectors per block varies depending on the disk drive being used; for the Atari 810 there are six sectors per block. A block is the smallest amount of disk space that can be allocated by DOS 4; thus, each file will always have a whole number of blocks allocated to it.

Each sector within a block is assigned an "offset": the first sector in a block has offset 0, the second sector in a block has offset 1, and so on. Similarly, the bytes within a sector are assigned "offsets": the first byte has offset 0, the second byte has offset 1, and so forth.

The allocation of blocks is controlled by the contents of a sector called the "volume table of contents" or "VTOC" (on some single-density disks the VTOC actually occupies two sectors). The bytes within the VTOC are assigned the following functions:

bytes	function
0	Mode identification code
1	Garbage list head pointer
2	Temporary use list head pointer
3	Garbage list block count
4	Zero
5	Temporary use list block count
6-7	Zero
8-$7F	Block lists
$80	Checksum
$81-$FF	Block lists (continued)

Each block on the disk has one byte in the VTOC associated with it. Block number 8 is associated with VTOC byte 8, block number 9 is associated with VTOC byte 9, and so on. The blocks are organized into lists, with each block's VTOC byte containing the number of the next block in the list.

Each file's blocks are organized into a list in the order in which the blocks occur within the file. The VTOC byte corresponding to the last block in the file contains the offset of the sector within which the last byte of the file is located.

The blocks not currently allocated to any file are organized into a list called the "garbage list". This list is always arranged in increasing numerical order. The last byte in the list contains 0.

Any time that DOS 4 allocates a block to a file, it will take the first block in the garbage list.

Blocks which have been allocated to files which are currently open for output are organized into a list called the "temporary use list". The temporary use list also contains any new blocks that have been allocated to files which are currently open for append.

The temporary use list begins with the blocks allocated to the file which has most recently been opened for output or append, arranged in the order that they occur within the file; next comes the blocks allocated to the second most recently opened output or append file; and so on. The last byte in the list contains 0.

The function of the temporary use list is to enable DOS 4 to recover blocks that are allocated to an output or append file which is never closed. Every time DOS 4 reads the VTOC, it checks to see if there are any blocks in the temporary use list; if there are, DOS 4 adds them to the garbage list.

Four bytes in the VTOC are allocated to hold the head pointer (number of the first block in the list) and block count (total number of blocks in the list) for both the garbage list and the temporary use list. The block counts are redundant information. Every time DOS 4 reads the VTOC it compares the block counts with the actual number of blocks in the list; if they disagree then DOS 4 will return ERROR 164.

The checksum byte in the VTOC is computed by the following program:

```
        LDY #0
        LDA #0
LOOP    CLC
        ADC (VTOC),Y
        ADC #0
        INY
        BPL LOOP
        STA (VTOC),Y
```

The use of VTOC byte $80 to hold a checksum does not leave a "hole" on the disk, because block number $81 begins immediately after the end of block number $7F.

Each entry in the disk directory occupies sixteen bytes, so that each directory sector holds eight entries on a single-density disk or sixteen entries on a double-density disk. The contents of a directory entry is:

bytes	contents
0	Flags: $00 = entry never used $40 = normal closed file $60 = file locked $80 = file deleted $81 = file open output
1	File list block count
2	Offset of last byte in file
3	File list head pointer
4	Zero
5-$C	Primary filename
$D-$F	Extender

The file list block count is redundant information. Any time that DOS 4 is asked to open, rename, delete, lock, or unlock a file, it will compare the block count in the directory entry to the actual number of blocks in the list. If the two do not agree, DOS 4 will return ERROR 164. This comparison is also made whenever DOS 4 is asked to close a file which was opened for append.

DISK DRIVE CONFIGURATION DATA

This chapter describes the memory locations which are used to configure the FMS to work with different kinds of disk drives. In general, application programs should not make use of these memory locations. Programs which absolutely must use these locations should be prepared to read Disk Configuration Files (or the power-on disk configuration program CONFIG.SYS, which contains copies of all the Disk Configuration Files) to obtain the data which is to be stored into these locations.

Each of the eight physical drives has associated with it one byte in the table MAPOFF and sixteen bytes in the table CONTYP. The FMS uses these tables to determine how it should access each drive in the system.

The symbol MAPOFF is defined to be $748 but actually refers to memory locations $749-$750. The contents of location MAPOFF+N, where $1 \leq N \leq 8$, is:

Bit 7 = 0 if physical drive N is double-density (i.e., has 256-byte sectors), and = 1 if physical drive N is single-density (i.e., has 128-byte sectors).

Bits 6-4 = Number of files currently open to physical drive N. You may examine these bits but you may not change them.

Bits 3-0 = Offset of the sector buffer which contains this drive's VTOC. The sector buffer at the lowest address in memory has offset 0, the buffer at the second lowest address has offset 1, and so on. You may examine these bits but you may not change them.

The table CONTYP is at [$751,$80]. The first sixteen bytes of CONTYP refer to physical drive #1, the second sixteen bytes refer to physical drive #2, and so on. The following describes the meaning of each byte in CONTYP.

DENSTY at [$751,1] contains the following:

Bit 7 = 1 if the drive returns an Atari 810 compatible data frame in response to a format command, and = 0 if it does not. (An "Atari 810 compatible data frame" is a data frame in which the first two bytes contain $FFFF if and only if there were no bad sectors detected during the format process.) If this bit equals 0 then the drive is still expected to return a data frame in response to a format command, but the contents of the data frame will be ignored. In any case, a

single-density drive is expected to return a 128-byte data frame and a double-density drive is expected to return a 256-byte data frame.

Bit 6 = 1 if the VTOC occupies two sectors, and = 0 if the VTOC occupies one sector. Two-sector VTOC's can be used only with single-density drives.

Bit 5 = 0.

Bits 4-0 = the current mode of this drive. Drive modes correspond to the selections on the Disk Utility Package's disk drive configuration menu. Mode 0 is choice A on the menu, mode 1 is choice B on the menu, and so on.

SCPERB at [$752,1] contains the number of disk sectors per block.

VTCSEC at [$753,2] contains the disk sector in which the VTOC is located. If the VTOC is two sectors long, then the contents of VTCSEC is the disk sector which contains the first sector of the VTOC; the next disk sector contains the second sector of the VTOC.

FSTSEC at [$755,2] is the first sector on the disk. For single-density disks it is normally 1. For double-density disks it is normally made greater than 1 so as to reserve space at the start of the disk for a boot file.

LOBLK at [$757,1] is the number of the first block on the disk. The first sector of the first block is the sector specified in FSTSEC. Note that the contents of LOBLK must be greater than or equal to the contents of SCPERB.

HIBLK at [$758,1] is the number of the last block on the disk. Note that on a single-density disk with only one VTOC sector, the contents of HIBLK may not exceed $7F. Also note that because byte $80 of the VTOC is used as a checksum, block number $81 follows immediately after block number $7F.

DIRBLK at [$759,1] contains the number of the block where the directory begins. The directory begins with the first sector in the block specified in DIRBLK and continues through sequentially numbered sectors.

DIRSEC at [$75A,1] contains the number of sectors in the directory. On a single-density disk, DIRSEC may not exceed 16 (decimal). On a double-density disk, DIRSEC may not exceed 8. (In other words, the directory cannot contain more than 128 entries.)

DIRCNT at [$75B,1] contains the number of blocks that are to be set aside for the directory and VTOC. DOS 4 will set aside sequentially numbered blocks beginning with the block specified in DIRBLK. (Note: If the directory is to include both block $7F and block $81 then DIRCNT must be one

greater than the number of blocks which are to be set aside for the directory and VTOC. This is to allow for the nonexistent block $80.)

DSPERB at [$75C,1] contains the number of sectors per block that DOS 4 is to use in calculating the sector counts which appear in the disk directory. If the disk has fewer than one thousand data sectors then normally DSPERB is given the same value as SCPERB. However, if the disk has one thousand or more data sectors then the value of DSPERB must be reduced so that the product of the value of DSPERB times the number of data blocks on the disk is less than one thousand. In any case, the value of DSPERB may not exceed 10 (decimal).

FMCOMD at [$75D,1] contains the serial or parallel bus command used to format the disk.

MODEID at [$75E,1] contains the disk mode identification code. When a disk is formatted, the value of MODEID is written into Byte 0 of the VTOC. Whenever the VTOC is read, the contents of Byte 0 is compared to MODEID and, if the two do not agree, ERROR 164 is generated.

DRVRES at [$75F,1] is reserved for future use and should be zero.

UNIT at [$760,1] contains the following:

Bit 7 = 1 for PIO bypass. If this bit is 1 then all I/O requests to the drive will be sent directly to the Serial I/O system, bypassing the Parallel I/O system. Note that it is illegal to set this bit to 1 when using the Revision A or B Operating System.

Bits 6-4 = 0

Bits 3-0 = Unit number. This number is placed into the DUNIT byte of the Device Control Block prior to calling SIO or PIO.

It is illegal to set UNIT in such a way that two different physical drive numbers refer to the same piece of hardware. (Note that it <u>is</u> legal to have two different <u>logical</u> drive numbers refer to the same piece of hardware. Also, it <u>is</u> legal to have, for example, one drive be unit #1 on the parallel bus and another drive be unit #1 on the serial bus.)

Certain conventions have been developed regarding the data stored in CONTYP:

1. Block numbering should be chosen so that block $7F is the last block on the first side of the disk and so that no block has a number less than 8.

2. The VTOC should occupy the last sector (or last two sectors) of the block(s) reserved for the

directory.

3. There should be enough unused space within the block(s) reserved for the directory to hold a second copy of the VTOC. (The current version of DOS 4 does not create a second copy of the VTOC but a future version may.)

4. The disk mode identification code (MODEID) is normally given a value of $53 for single-sided disks, or $44 for double-sided disks.

DISK CONFIGURATION FILE FORMAT

This chapter describes the standard Disk Configuration File format used by DOS 4. In order to use the information in this chapter, you will need to have a thorough understanding of the chapters on DOS 4 Disk Drive Configuration Data and DOS 4 Disk File Structure.

GENERAL INFORMATION

A Disk Configuration File (DCF) is a file which contains a description of a disk drive. This description provides DOS 4 with all the information that it needs to have in order to work on that drive.

Each DCF describes one disk drive model. (In point of fact, a single DCF can usually handle a family of two or more different but related disk drives; for our purposes we will consider such a family of disk drives to be one "model".) The user must obtain a DCF for each model of disk drive that he has and merge each of them into his copy of the DOS 4 Disk Utility Package.

Each disk drive model has one, two, or three modes into which it can be configured (for example: single side/single density, single side/double density, and double side/double density). When three different modes are provided, they should be as follows:

1. One mode which is the "standard" mode for the given disk size. For example, for 5¼-inch minifloppy disk drives, the "standard" mode is a mode that can read Atari 810-format diskettes. It is intended that all commercially available software be distributed in the "standard" mode.

2. One mode which uses only the first side of the disk and offers the maximum possible storage capacity.

3. One mode which uses both sides of the disk and offers the maximum possible storage capacity.

There is enough room in the DUP's menu of disk drive configurations to support 8 different disk drive models. Since the DUP is supplied with the Atari 810 DCF already merged into it, this means that up to 7 DCF's can be merged into the DUP by the user. The total amount of memory available in the DUP for DCF's is a little more than 4K. Therefore, individual DCF's should not be much bigger than ½K.

OUTLINE

Each DCF consists of seven components. In this section, we will list the seven components and describe each of them briefly. In the remainder of this chapter, we will describe each component in detail. The seven components must occur in the same order that they are listed here. The seven components are:

1. A 5-byte header which contains a file identification code, the length of the file, and the number of different modes.

2. Two bytes of title information which specifies the location and length of the title.

3. Configuration data, 19 bytes for each mode. Most of this is just copied into the FMS's disk drive configuration data tables.

4. A 4-byte vector table which specifies the starting locations of the device-dependent routines.

5. The title, a character string which specifies how this disk drive model is to be listed in the menu of disk drive configurations.

6. The configuration routine, a subroutine which performs the device-dependent part of configuring the drive.

7. The write-boot routine, a subroutine which writes a boot file to the disk.

1. THE HEADER

Each DCF must begin with a 5-byte header. The first two bytes of the header contain the file identification code, which must be $FEFE. The function of the file identification code is to enable the DUP to tell the difference between DCF's and other types of files.

The third and fourth bytes of the header contain the total length of the DCF, not counting the 5-byte header itself. The length is stored low byte first.

The fifth byte of the header contains the number of modes. It must be between 1 and 3 (inclusive).

2. TITLE INFORMATION

Following the header comes two bytes of title information. The first byte contains the address of the first byte of the title minus the address of the first byte of title information. The second byte contains the total number of characters in the title (counting the carriage returns).

3. CONFIGURATION DATA

This component of the DCF contains a 19-byte data table for each mode. Thus, the configuration data is 19 bytes long if there is only one mode, 38 bytes long if there are two modes, and 57 bytes long if there are three modes.

When the DUP constructs its menu of disk drive configurations, these modes will be assigned consecutive letters. The mode whose table appears first in this section will recieve the first letter, the mode whose table appears second will receive the second letter, and the mode whose table appears third will receive the third letter.

The order in which the three modes are listed must be carefully chosen in order to make the DUP's IDENTIFY MODE command function accurately. The IDENTIFY MODE command works as follows: For each of the three modes, beginning with the last mode and working toward the first mode, the DUP attempts to (a) configure the drive into the mode in question, (b) read the last data sector on the disk, and then (c) read the VTOC and verify that the mode identification code, garbage list block count, and temporary use list block count are correct. If all three tests succeed, the IDENTIFY MODE operation terminates and the mode in question is displayed. If any of the tests fail, the DUP goes on to the next mode. When all the modes have been tried unsuccessfully, the message UNABLE TO IDENTIFY MODE is displayed.

Each 19-byte table has the following form:

Byte 0: Bit 7 = 1 if the drive returns an Atari 810 compatible data frame in response to a format command. Bit 6 = 1 if the VTOC is two sectors long. Bit 5 = 0. Bits 4-0 are ignored. (Bits 7-5 of this byte are copied into bits 7-5 of DENSTY. Bits 4-0 of DENSTY are supplied by the DUP.)

Byte 1: Number of sectors per block. (This byte is copied into SCPERB.)

Bytes 2-3: Disk sector which contains the VTOC. (These two bytes are copied into VTCSEC.)

Bytes 4-5: The first sector on the disk. (These two bytes are copied into FSTSEC.)

Byte 6: The first block on the disk. (This byte is copied into LOBLK.)

Byte 7: The last block on the disk. (This byte is copied into HIBLK.)

Byte 8: The block where the directory begins. (This byte is copied into DIRBLK.)

Byte 9: The number of sectors in the directory. (This byte is copied into DIRSEC.)

Byte 10: The number of blocks in the directory. (This byte is copied into DIRCNT.)

Byte 11: The number of logical sectors per block. (This byte is copied into DSPERB.)

Byte 12: The serial or parallel bus format command. (This byte is copied into FMCOMD.)

Byte 13: Disk mode identification code. (This byte is copied into MODEID.)

Byte 14: Zero. (This byte is copied into DRVRES.)

Byte 15: Contains $80 if sectors are 128 bytes long. Contains 0 if sectors are 256 bytes long. (This byte is used to set bit 7 of MAPOFF.)

Byte 16: Contains the address of this mode's "mode star" minus the address of byte 0 of this data table. Refer to the description of the title for information about "mode stars".

Byte 17: Contains the mode-dependent argument. Whenever the DUP calls one of the device-dependent routines (the configuration routine or the write-boot routine), the mode-dependent argument for the drive's current mode is passed to the routine in the 6502 Accumulator. The mode-dependent argument is not otherwise used by DOS 4.

Byte 18: Contains the address of the first byte of the vector table minus the address of byte 0 of this data table.

4. THE VECTOR TABLE

Following the configuration data is a 4-byte vector table which specifies the starting addresses of the two device-dependent routines.

The first and second bytes of the vector table contain the address of the first byte of the configuration routine minus the address of the first byte of the vector table.

The third and fourth bytes of the vector table contain the address of the first byte of the write-boot routine minus the address of the third byte of the vector table.

5. THE TITLE

The title is a character string which specifies how this drive model is to be listed in the DUP's menu of disk drive configurations. The title should consist of two lines of text, each terminated with a carriage return. A line of text cannot contain more than 38 characters (counting the carriage return).

The title should contain only standard ASCII characters — no inverse-video, graphics, or cursor control characters. Uppercase letters are preferred.

Each mode is associated with a two-character substring of the title. The first character of this substring is called the "mode star", and the second character is called the "mode letter". When the menu of disk drive configurations is displayed, these substrings will be replaced with characters supplied by the DUP. The mode star will be replaced with an asterisk if the drive is currently configured to the mode in question, and with a blank otherwise. The mode letter will be replaced with an uppercase letter from A to X which represents the key that the user must press to select the mode.

6. THE CONFIGURATION ROUTINE

The configuration routine is a subroutine which performs the device-dependent part of the disk drive configuration process. Whenever the DUP wants to change the configuration of a disk drive, it first copies the disk drive configuration data for the desired new mode from the table in the DCF to the File Management System's CONTYP and MAPOFF tables. After the data has been copied, the DUP calls the configuration routine in the DCF.

The DUP can change the configuration of a disk drive when the user selects the CONFIGURE DRIVE command, or when the user selects the /R option of the DUPLICATE FILE or DUPLICATE DISK commands. The disk configuration may also be changed by the power-up disk configuration program CONFIG.SYS (the CONFIG.SYS file is just a copy of the part of the DUP that does disk configurations).

The normal function of the configuration routine is to send serial or parallel bus commands to the drive that tell the drive how it should configure itself. If the drive does not require any such commands, then the configuration routine should at least send out a STATUS command to verify that the drive is really there. The configuration routine is also allowed to modify the CONTYP and MAPOFF tables, if desired.

The configuration routine must obey the rules described the section entitled "Rules For Device-Dependent Routines" (below).

7. THE WRITE-BOOT ROUTINE

The write-boot routine is a subroutine which performs the device-dependent part of the process of making bootable disks. The DUP will call the write-boot routine for each destination disk of a WRITE DOS command or DUPLICATE DISK command.

The normal function of the write-boot routine is to write an invisible boot file to the first few sectors of the disk. If the drive does not require a boot file, then the write-boot routine should just return without doing anything.

The boot file normally contains a program which configures the disk drive and then loads and runs the File Management System. Thus, the write-boot component of the DCF actually contains two programs: one program to write the boot file, and a second program which is written into the boot file.

The program which writes the boot file must obey the rules described in the section entitled "Rules For Device-Dependent Routines" (below). The program may examine the CONTYP and MAPOFF tables, but is not allowed to modify them. Also, the program is not allowed to either examine or modify any sectors on the disk except those that are specifically reserved for the boot file (by specifying a value for FSTSEC that is greater than one).

The program which is written into the boot file should obey the following rules:

1. The boot program should assume that the FMS begins in the first data sector on the disk (the sector identified in FSTSEC) and continues through sequentially numbered sectors. (Exception: if the disk directory is located at or near the start of the disk then the FMS will hop over the directory.) In particular, the boot program need not search the disk directory or read the VTOC. The FMS is always loaded from disk drive unit #1.

2. The boot program may assume that the FMS loads into a contiguous block of memory beginning at location $701. The second byte of the FMS (which loads at location $702) contains the number of 128-byte blocks that are to be loaded. It is guaranteed that there will not be more than 34 such blocks.

3. If the boot program is unable to successfully configure the drive and load the FMS, then it should set the 6502 carry bit (with SEC) and return (with RTS).

4. After the FMS is successfully loaded, the boot program must copy the contents of memory

locations $705-$706 into locations $0C-$0D and then JMP to location $707.

RULES FOR DEVICE-DEPENDENT ROUTINES

The device-dependent disk drive configuration routine and the device-dependent write-boot routine must obey the following rules:

1. The routine must be relocatable.

2. At the time the routine is called, memory location DSKUTL at [$1A,2] contains the address of the first byte of the routine. (In fact, the DUP transfers control to the routine with a JMP (DSKUTL) instruction.) The routine is allowed to modify DSKUTL.

3. The routine must return by loading an error status code into the 6502 Y-register and then executing an RTS instruction. A status code greater than or equal to $80 will cause an error message to be generated. It is not necessary to set the 6502 sign bit to reflect the value of the status code.

4. Memory locations $19FC through $1BFB (inclusive) are available for use as scratch storage.

5. The routine may communicate with the disk drive by calling OS entry point SIOV.

6. When the routine is called, OS variables DDEVIC, DTIMLO, DAUX1, and DAUX2 contain $31, 7, 0, and 0, respectively.

7. When the routine is called, OS variable DUNIT contains the unit number of the disk drive in question.

8. When the routine is called, if PIO bypass has been selected for the disk drive in question, then OS variable PDVMSK contains 0. This forces I/O requests to go to the serial bus instead of the parallel bus. The DUP will automatically restore PDVMSK to its original value after the routine returns.

9. When the routine is called, the 6502 X-register contains the offset into CONTYP for the disk drive in question, i.e., the X-register contains 16*(PDN-1) where PDN is the physical drive number.

10. When the routine is called, the 6502 Accumulator contains the mode-dependent argument obtained from byte 17 of the configuration data table in the DCF. The configuration routine should use the value of the Accumulator to determine which mode the drive is to become. The write-boot routine should use the value of the Accumulator to determine which mode the drive is in.

11. When the routine is called, the 6502 Y-register contains the value 1, and the 6502 sign bit is clear. This means that a device-dependent routine which does nothing can consist of just an RTS instruction, and that a routine can jump over an initial data table by means of a BPL instruction.

12. The routine should not communicate directly with the user.

SYSTEM MEMORY LOCATIONS

This chapter describes fixed memory locations in DOS 4 which are intended primarily to allow the various programs that comprise DOS 4 to communicate with each other. These memory locations should not be of interest to application programs. While it is unlikely that any of these locations will be changed in a future version of DOS 4, this is not guaranteed.

COMMAND PROCESSOR / DISK UTILITY PACKAGE LINKAGE

Linkage between the CP and DUP is controlled by three memory locations within the Command Processor: VMENU at [$1BFC,2], VMENLO at [$1BFE,2], and VMENHI at [$1C00,2].

The DUP uses the contents of these locations when it writes out the CP during the WRITE DOS FILES command. The CP consists of an initialization section followed by a run section. The initialization section extends from address $1BFC to one less than the address contained in VMENLO, and executes beginning at address $1C02. The run section extends from the address contained in VMENLO to one less than the address contained in VMENHI, and executes beginning at the address contained in VMENU.

The DUP also uses VMENU to implement the COM PROCESSOR command. In general, any program that does not overwrite the CP can return to the CP by a JMP (VMENU) instruction; this avoids the overhead of re-loading the CP.

DISK UTILITY PACKAGE / DISK CONFIGURATION FILE LINKAGE

Linkage between the DUP and DCF's is controlled by five memory locations within the Disk Utility Package: UTLTOP at [$2103,2], MODNUM at [$2105,1], MDLNUM at [$2106,1], MODTAB at [$2107,$30], and MDLTAB at [$2137,$10]. These memory locations are also present within the power-on disk configuration program CONFIG.SYS.

UTLTOP contains the address of the first free byte above the top of the DUP. Whenever the DUP merges a DCF into itself, it reads the 5-byte DCF header into a temporary location, then reads the rest of the DCF into memory beginning at the address contained in UTLTOP, and finally adds the length of the DCF to UTLTOP (the DCF header is discarded).

MDLNUM contains the number of disk drive models that have been merged into the DUP; it may

not exceed 8. MODNUM contains the total number of different disk drive modes that are supported; it may not exceed 24 (thus allowing a maximum of 3 modes per model).

MDLTAB and MODTAB are tables of addresses, with each address occupying two consecutive bytes and with each table being filled starting at the lowest address. MDLTAB contains the address where each model's DCF begins; this is the address that was in UTLTOP when the DCF was merged. MODTAB contains the address of the first byte of each mode's 19-byte DCF configuration data table; these addresses are calculated when the DCF's are merged.

POWER-ON DISK CONFIGURATION PROGRAM LINKAGES

Memory location AUTSPC at [$712,$10] within the FMS contains the name of the file which contains the power-on disk configuration program, normally D1:CONFIG.SYS. The FMS will load and run this program during coldstart. If the program is not present, or if the load cannot be done successfully, then the system will (by design) freeze.

Location AUTOSP at [$2147,$10], which is present in both the DUP and the power-on disk configuration program, contains the name of the file to which the power-on disk configuration program will attempt to link after the disk drives are configured. This is normally D1:AUTORUN.SYS.

SDTYPE at [$73E,1] within the FMS is a shadow for DTYPE+1. At the start of the WRITE DOS command, the Disk Utility Package copies DTYPE+1 into SDTYPE and then stores a 1 into the four low-order bits of DTYPE+1 (the original value of DTYPE+1 is restored at the end of the WRITE DOS command). This has the effect of forcing the power-on disk configuration program D1:CONFIG.SYS and the power-on application program D1:AUTORUN.SYS to load from physical drive #1, even if logical drive D1: has been redirected to some other physical drive. The power-on disk configuration program copies the four low-order bits of SDTYPE into DTYPE+1 after it has opened (or attempted to open) the power-on application program.

FILE MANAGEMENT SYSTEM / BOOT LOADER LINKAGE

Linkage between the FMS and the boot loader is controlled by memory locations $701-$709. Location [$701,1] contains a flag byte whose value is 0; it has no function. Location [$702,1] contains the number of 128-byte blocks which must be read from the disk in order to load the FMS. Location [$703,2] contains the origin address of the FMS, which is $701.

Location [$705,2] contains the initialization address of the FMS. This address is copied into OS variable DOSINI by the boot loader. It is possible to re-initialize the FMS without re-initializing other programs that are hooked into the DOSINI chain by storing a nonzero value into the OS variable WARMST and then making a subroutine call to a JMP ($705) instruction.

Location [$707,3] contains a JMP instruction to the FMS's boot continuation routine. The boot loader terminates with a JMP $707 instruction, allowing the FMS to complete the boot load process.