SHEPARDSON
MICROSYSTEMS INCORPORATED

OPTIMIZED SYSTEMS SOFTWARE

CP/A OS DFM

Control Program/Apple,
Operating System, Disk File Manager

OPTIMIZED SYSTEMS SOFTWARE

Control Program/Apple

for the Apple II (R)

Feb 1980

Version 1.0

Optimized Systems Software
Shepardson Microsystems, Inc.
20823 Stevens Creek Blvd, Bldg C4-H
Cupertino, CA 95014
Telephone: 408-257-9900

TABLE OF CONTENTS

GENERAL INFORMATION

CP/A is a general purpose command control program for the OSS disk operating system. The CP/A user has two types of commands available: intrinsic commands and extrinsic commands. The intrinsic commands are those commands which are executed directly via CP/A code. Extrinsic commands are executed by loading and running a program.

The following commands are CP/A intrinsic.

DIRECTORY	List directory
SAVE	Save a program
LOAD	Load a program
RUN	Run a program already in RAM
ERASE	Erase a file from it's medium
PROTECT	Protect a file from erasure or change
UNPROTECT	Unprotect a protected file
RENAME	Rename a file

The CP/A examines the first three characters of the user input for a match with the intrinsic commands. If the first three characters match, the remaining contiguous characters through a blank ($20), carriage return ($0D) or a comma are ignored. Thus DIR, DIRECTORY, DIRGLOP, etc all access the DIRECTORY intrinsic command.

The CP/A, upon determining a command is not intrinsic, will attempt to execute an extrinsic command. The user command is converted into a filespec of the form:

Device: command.COM

The device is a single character device specifier (usually A or B). If the user does not specify a device, then the default device is used. The .COM is always appended to the command. The command BASIC will generate a filespec of:

A:BASIC.COM

CP/A will next attempt to open and load a file with the generated filespec. If the file load is properly terminated, CP/A will transfer to the loaded program's start location. The user loaded program now has control of the system.

DEFAULT DEVICE

CP/A starts execution with a default device of "A:" which is disk drive 1. The user may change the default drive by entering the new drive spec (ie "B:") followed by a carriage return. CP/A uses the default drive as the input line prompt character in the form: "A."

The default device is used by CP/A in all cases where it constructs a filename from user input and the user has not specified a device. The command

```
LOAD    BASIC.COM
```

will load A:BASIC.COM assuming the default drive is "A:". The command

```
LOAD    B:BASIC.COM
```

will load B:BASIC.COM no matter what the default drive is.

COMMAND DETAILS

For each command, the command syntax is followed by an example of actual usage and a decription of the command's operation.

SAVE

```
SAV[E]  filespec        start-hex-adr   end-hex-adr

SAVE    TEST            800             B00
```

A file will be created with the name "filespec" and will contain all data from "start-hex-adr" up to, but not including "end-hex-adr". CP/A will write a four byte save file header before the data. The first two bytes are "start-hex-adr" and the second two bytes are "end-hex-adr". This four byte header is compatible with the OSS Assembler object output.

LOAD

```
LOA[D]  filespec

LOAD    TEST
```

The specified file will be loaded. The files first four bytes are used to determine the load start address and end address. The start address must be less than the end address. The file load start address is placed into the OSS go-location (3F9).

RUN

```
RUN     optional-hex-adr

RUN     $800
```

CP/A will branch to the run address. The address is either the specified hex address or, if unspecified, the address at the go-location. The address at the go-location is set by system initialization (to CP/A), the act of LOADing a program, or by an application program that has called CP/A. BASIC and the ASSEMBLER both set the go-location at their respective warmstart entry points.

DIRECTORY

```
DIR[ECTORY]      optional-filespec
DIR              *.COM
```

The CP/A will open the specified "device:filespec" for directory listing. If the user does not specify a filespec, the default

```
default-device: *.*
```

filespec will be used.

ERASE

```
ERA[SE]          filespec

ERASE            TEST.*
```

The specified file or files will be erased from the device, provided that they are not protected.

PROTECT

```
PRO[TECT]        filespec

PRO              BASIC.COM
```

The specified file or files, are protected from modification, erasure or renaming.

UNPROTECT

```
UNP[ROTECT]      filespec

UNP              DATA.TST
```

The specified file or files, are unprotected. They may now be erased, modified, or renamed.

RENAME

```
REN[AME]         old-filespec     new-filespec

REN              GLOP             ACTS.NEW
```

The files matching the old-filespec are renamed according to the new-filespec.

INIT (Extrinsic command)

INIT (no parameters)

The INIT command is used to physically and logically initilize an OSS format diskette. A diskette can be used by the OSS system (version 1) if and only if it has been initialized by INIT. Initializing a diskette destroys all previous information on the diskette.

The INIT program (INIT.COM) begins by requesting the INIT function to be performed. These functions are

1) INIT a disk with boot record.
2) INIT a disk without boot record.
3) Re-write the boot record.
4) Return to CP/A.

The first two functions physically and logically initialize the working surface of the disk. If a disk is initialized without a boot record, the disk will contain 719 sectors of 128 bytes each; however, the diskette is not bootable. If the disk is initialized with a boot record, the disk will contain 680 sectors of 128 bytes each and a 6.5K boot record. The boot record contains the operating system located from page $A8 thru page $BF plus two additional pages of boot loader code. The third function is used to re-write the operating system to the boot record. The OSS disk must have been previously formatted using function 1 to execute function 3.

The OSS File Manager (Version 1) uses 9 sectors for file management information. The INIT program also logically formats these 9 sectors. (See OSS DFM document.)

DUPDSK (Extrinsic Command)

DUPDSK (no parameters)

The DUPDSK command is used to make a duplicate copy of one OSS disk on to another OSS disk. Both disks must be initialized in the same way; they both must either have or not have a boot record. The boot record is not copied by DUPDSK.

COPY (Extrinsic Command)

```
COPY    from-filespec     to-filespec
COPY    A:BASIC.COM       B:BASIC.COM
```

The from-file is copied to the to-file. The to-file does not have to be a disk file, but may be any device. The from-file is unmodified. The from-filespec need not be a disk file, but it must provide an EOF to terminate the copy.

USER WRITTEN EXTRINSIC COMMANDS

Any user file of the name "name.COM" may be used as a CP/A extrinsic command. The program may be placed at any memory location that does not interface with other concurrent programs. The program entry point will be at the address of the first byte SAVEd.

The ASCII command line that was entered to invoke the extrinsic command is placed by CP/A at location $280. The executing extrinsic program can examine this (unmodified) command line for parameters it may require. The current default device value is located at $BCFE in version 1 of the OSS system. CP/A will jump to the extrinsic command with IOCB numbers 1 through 7 closed. IOCB number 0 is open for the current console device for both input and output. IOCB number 0 should not be opened or closed by the command code (unless that is the purpose of the command). The normal command exit is to CP/A at location $BFFD.

OPTIMIZED SYSTEMS SOFTWARE

OSS OPERATING SYSTEM

for the Apple II (R)

Feb 1980

Version 1.0

Optimized Systems Software
Shepardson Microsystems, Inc.
20823 Stevens Creek Blvd, Bldg C4-H
Cupertino, CA 95014
Telephone: 408-257-9900

TABLE OF CONTENTS

GENERAL INFORMATION

The OSS Operating System provides the user with a uniform I/O interface to the various system I/O devices. The user places I/O command information in one of eight system I/O control blocks (IOCBs) and calls the OS entry point (OS in system memory map). The OS interperts the command and calls upon a Device Handler to perform the requested I/O operation. OS device handlers are coded in a specific format that provides a uniform interface to the OS.

The OS uses a file specifier to determine the device handler that is to be used for the I/O operation. The file specifier is an ASCII string of the format:

DN: filename

D - Device character code that indentifies that device handler in the device table. The D may be any ASCII value.

N - Optional sub device specifier. The N, if specified, must be an ASCII 0-9. The OS will supply a default value of 1.

filename - Optional filename. If the device handler requires a filename, it must directly follow the required colon. The filename format is set by the requirements of the device handler.

IOCBs

There are Eight IOCBs in the system. The first IOCB (IOCB #0) is located at address IOCB (see memory map). Each IOCB is 16 bytes in length and all IOCBs are contiguous. The following details the specific use of each IOCB byte.

FIELD	DISPL	LENGTH	
DHID	0	1	Device Handler Index. Set by OS. DHID is $FF if IOCB not open.
DVCNO	1	1	Device Handler sub-device number. The binary value ($00-$09) of the N in the file specifer. (Default = 1).
OSCOM	2	1	Operating System command. The command OS is to execute.
IOSTAT	3	1	I/O operation status. In general, values greater than or equal to 128 ($80) are errors.
BUFADR	4	2	User buffer adr in the normal 6502 low/high order. Points to File Specifer (when required), or to user data buffer.
PUTADR	6	2	The address (minus one) of the DH put routine. The user may call the DH put routine directly using this vector.
BUFLEN	8	2	User buffer length in the normal 6502 low/high order. If BUFADR points to File Specifier, then BUFLEN is not required.
AUX1	10	1	Auxillary Byte 1. This byte is used to contain the open type code while the IOCB is open.
AUX2	11	1	Auxillary bytes 2-6 used as required by individual Device Handlers.
AUX3	12	1	
AUX4	13	1	
AUX5	14	1	
AUX6	15	1	

OS COMMANDS

The OS commands fall into three general classes.

1) OPEN and CLOSE
The user specifed IOCB is opened for use with the device specified by the File Specifier.

The specified IOCB must not be currently opened. The OS will determine the requested device handler from the file specifier and will place the device handler index in the IOCB The device handler open routine will be called to provide whatever device open functions are required. Once the IOCB has been properly opened, it may be used for data I/O and Device Dependent commands.

When the user has finished with the Device, the IOCB should be closed via the OS CLOSE command.

2) DATA I/O
The OS performs I/O operations to and from a user record buffer. The user supplies the OS with the address of a buffer and a data buffer length. There are five types of DATA I/O commands. These commands will be detailed later in this document.

3) DEVICE DEPENDENT COMMANDS
Device Dependent Commands are those commands that are not universal to all devices, but are specific to a particular device. The OS interperts all commands above a certain value to be Device Dependent Commands. If the IOCB used, has not been opened, OS assumes that a filespec is present and acts upon it in the same manner as open; (except the DH open routine is not called and the IOCB is "open" for the one command only).

The following list details the OS commands and the data required for each command.

COMMAND	VALUE(HEX)	DESCRIPTION
OPEN	$01	Open a device for I/O. The address of the filespec is pointed to by BUFADR. AUX1 must have 04 bit on if input, 08 bit on for output, or both 04 and 08 bits on if device is to used for input and output. AUX1 may have other bits set on for special device handler OPEN functions.
GETRECORD	$04	A record of length BUFLEN will be moved into the buffer pointed to by BUFADR. The IOCB must have been OPENed for input.

GETLINE	$05	A line of ASCII input terminated by a carriage return ($0D) will be placed in a buffer pointed to by BUFADR. The BUFLEN field determines the maximum line size. The IOCB must have been OPENed for input.
PUTRECORD	$08	A record of length BUFLEN will be sent to the device from the buffer pointed to by BUFADR. The IOCB must have been OPENed for output.
PUTLINE	$09	A line of ASCII input terminated by a carriage return ($0D) will be sent to the device from the buffer pointed to by BUFADR. The IOCB. must have been OPENed for output.
CLOSE	$0C	The IOCB and file are closed.
STATUS	$0D	The device will return a status byte in the IOSTAT field. The status returned is Device Dependent. The IOCB need not have been OPENed. If not OPEN, BUFADR must point to a file specification.
DEVICE DEPENDENT	$0E-$7F	The DEVICE DEPENDENT commands are sent directly to the device handler. The IOCB need not have been OPENed. If not OPEN, BUFADR must point to a file specification. (The OSS Disk File Manager supports commands $20-$26; see the OSS DFM documentation for details.)

OS STATUS

All OS operations return a status value in the IOSTAT field. OS convention is that status values of $80 or greater indicate some sort of error.

VALUE(HEX)	MEANING
$01	No error or warning.
$02	Truncated ASCII line. The OS did not find a $0D within BUFLEN for ASCII line I/O.
$03	End of file look ahead. The last byte transfered from the DH was its end-of-file byte. The DH must set this status.
$80	Operation aborted. Set by Device Handler.
$81	Device not ready. Set by Device Handler.
$82	Device does not exist. The device was not found is the OS device table.
$83	Data Error. Set by Device Handler.
$84	Invalid Command. The Device Handler has rejected the command.
$85	Device/File not open. The IOCB has not been OPENed for the operation.
$86	The IOCB specified is invalid.
$87	The device is write protected.

Various Device Handlers may set other values as required.

USING THE OS from ASSEMBLY LANGUAGE

Once the user has set up an IOCB with the required information, the X-register is loaded with the IOCB number (0-7) times 16 and the OS is called at the OS entry point (see memory map). The OS will return to the user with X register unmodified, the Y register will contain the status value, and the accumulator value is unpredictable. The following is an example:

```
LDX     #$50            ; USING IOCB #5
JSR     OS              ; CALL OS
TYA                     ; SET PROCESSOR STATUS FLAG
BMI     ERROR           ; BRANCH IF ERROR
BPL     GOODIO          ; ELSE I/O WAS GOOD
```

DEVICE HANDLERS

A user may create a Device Handler for any required purpose. The user need only code the DH according to the OS conventions and make a unique entry for the device in the OS device table.

The Device Handler table contains eight possible entries. The OSS system as shipped uses four of the entries, the remaining four are avaiable to the user. The format of a Device Handler table entry is as follows.

FIELD	LENGTH	DESCRIPTION
DNAME	1	Device Name. Usually an ASCII value. OS uses A, B, E, and P. A zero DNAME indicates an unused entry.
DHVTA	2	Device Handler Vector Table Address. The address of the DH vector table in normal 6502 low/high fashion.

The Device Handler Vector Table contains six consecutive address (normal 6502 type) that point to the routines in the DH that perform the indicated functions.

1) Open Device
2) Get Device Status
3) Get Data Byte
4) Put Data Byte
5) Close Device
6) Device Dependent Command

The OS will call one of the six Device Handler functions directly via DHVT. Upon entry to the DH function the X register will contain the IOCB number (0-7) times 16. The user may use the register to directly access the specified IOCB via the abs,X instructions. When the Put Data Byte function is called, the accumulator will contain the data byte. The Device Handler must return a status value to OS in the Y register. If the Get Byte function is called, the data will be returned in the accumulator.

The zero page locations DHZPG through DHZPGE (see memory map) are available for use by device handlers as temporary storage. These locations are subject to change upon exiting from the DH code.

DEVICE E:

The device E: (EDITOR) is a device handler which interfaces to the Apple Monitor "getline" and "putline" routines. The E: device handler provides the user with all the line editing features provided by whatever Apple Monitor prom the user has installed. All E: I/O is accomplished through the output vector routine at $36 and the input vector routine at $38. The vectors are initialized by OSS to use the Apple Keyboard and CRT screen.

IOCB #0 is used by OSS as the system console and is opened using device E: upon system initialzation. All OSS system programs (CP/A, BASIC, DMGR, etc) use IOCB #0 for console I/O. No OSS system routine closes IOCB #0.

The user may change the console device from the Apple keyboard and screen. There are two ways of accomplishing this. The vectors at $36 and/or $38 may be modified, or IOCB #0 may be closed and reopened to another device. The first method will retain the Apple monitor line edit features such as backspace and line delete. The second method will provide line editing if and only if the device handler used provides for line editing.

See Appendix A for listing of Device E: routine.

DEVICE Pn:

The device Pn: (PORT n) is a device handler for the eight Apple slots. The "n" specifies which port is to be used (0-7).

When a port is OPENed the device address (C100, C200 etc) of the port is stored in the IOCB. When a Pn: data byte I/O is called for, the following sequence occurs:

1) The device address saved in the IOCB are swapped with the vectors at $36 and $38.

2) If the function is PUTBYTE, the most significant bit ($80) of the data byte is inverted and the byte is output through location $36. If the function is GETBYTE the data byte is obtained through location $38. The received data byte's most significant bit ($80) will be inverted by th Pn: device handler.

3) The device address at $36 and $38 will be swapped with the device address in the IOCB.

The sequence of operations of Pn: allow the user to open several ports simultaneously and perform I/O through them as required. The inversion of the data byte's most significant bit is required because all OSS software is ASCII based.

See Appendix A for listing of Device P: routine.

SYSTEM MEMORY MAP (Version 1.0)

LOCATION	LABEL	USAGE
BFFD	CPRTN	JMP CP/A
BFFA	SINIT	JMP system initilizer
BFF8	HIMEM	HIMEM
BFF6	LOMEM	LOMEM
BFF5	OSVER	OSS version number
BDA0	OSENT	OS entry address
BD87	DHTAB	Device table (8 devices)
BD80	DIOB	Disk I/O Block
BD00	IOCB	IOCBs (8 IOCBs)
BCFE	DEFDRV	Default Drive (ASCII character)
B900	CPAENT	CP/A entry address
B850		E: and P: device handlers
ADA0	DFMNUMF	Number of file buffers (4 default)
ADA1	DFMDIR	File buffer allocation direction ($80)
ADA2	DFMBUF	File buffers start address ($AB00)
AB00	DIOENT	Disk I/O Routine
A800		File buffers
0800		User Ram
0400		Apple screen buffer
03F9		JMP go-location
03F0		Auto start Rom vectors
0280	CMDLINE	CP/A command line
0200		Line buffer and work space
0100		6502 stack
0080		Application zero page Ram
007F	DHZPGE	Top of Device Handlers temps
0079	DHZPG	Start of Device Handlers temps
0068		OSS system zero page
0050		Available zero page
0020		Apple Monitor Ram
0000		Available zero page

APPENDIX A

```
                                PAGE    'PORT DEVICE HANDLER'
580                     ;
581                     ;       APPLE PORT DEVICE
582                     ;
583     007C    PDHCHR  EQU     AEDCHAR         ; DATA CHAR
584     007A    PDHICD  EQU     DHZPG+1         ; IOCB DISPL
585     007B    PDHFLG  EQU     DHZPG+2         ; I/O FLAG
586     BFFD            ORG     $B850
587                     ;
588     B850    PORTDH  EQU     *
589 B850 5CB8           DB      @@PDHOPN        ; OPEN
590 B852 C5B8           DB      @@AEDSTA        ; STATUS
591 B854 79B8           DB      @@PDHGBT        ; GET BYTE
592 B856 73B8           DB      @@PDHPBT        ; PUT BYTE
593 B858 C5B8           DB      @@AEDSTA        ; CLOSE
594 B85A C5B8           DB      @@AEDSTA        ; DEVICE DEPENDENT
595                     ;
596     B85C    PDHOPN  EQU     *               ; OPEN PORT N
597 B85C A900           LDA     #0              ; SET ZERO TO
598 B85E 9D0FBD         STA     ICAUX6,X        ; LOW ADDR BYTE OUT
599 B861 9D0DBD         STA     ICAUX4,X        ; LOW ADDR BYTE IN
600                     ;
601 B864 BD01BD         LDA     ICDNO,X         ; GET DEVICE NO
602 B867 2907           AND     #$07            ; ISOLATE 3 LSB
603 B869 09C0           ORA     #$C0            ; OR IN ADR HI
604 B86B 9D0EBD         STA     ICAUX5,X        ; HIGH ADDR BYTE OUT
605 B86E 9D0CBD         STA     ICAUX3,X        ; HIGH ADDR BYTE IN
606 B871 D052           BNE     AEDSTA          ; DONE
607                     ;
608     B873    PDHPBT  EQU     *               ; PUT DATA BYTE
609 B873 857C           STA     PDHCHR          ; SAVE DATA BYTE
610 B875 A900           LDA     #0              ; INDICATE OUTPUT
611 B877 F002           BEQ     PDHGP
612                     ;
613     B879    PDHGBT  EQU     *
614 B879 A9FF           LDA     #$FF            ; INDICATE INPUT
615                     ;
616     B87B    PDHGP   EQU     *
617 B87B 857B           STA     PDHFLG          ; SAVE FLAG
618 B87D 8A             TXA
619 B87E A8             TAY                     ; IOCB DISPL TO Y
620 B87F A203           LDX     #3              ; SAVE X FOR 4 BYTE MOVE
621                     ;
622 B881 B536   PDHM1   LDA     $36,X           ; GET APPLE SWITCH BYTE
623 B883 48             PHA                     ; SAVE ON STACK
624 B884 B90CBD         LDA     ICAUX3,Y        ; MOVE VECTOR FROM IOCB
625 B887 9536           STA     $36,X           ; TO APPLE SWITCH BYTE
626 B889 C8             INY
627 B88A CA             DEX
628 B88B 10F4           BPL     PDHM1           ; BR IF MORE TO MOVE
629                     ;
630 B88D 847A           STY     PDHICD          ; SAVE IOCB DISPL
631 B88F 20A6B8         JSR     PDHGO           ; GO DO I/O
632 B892 857C           STA     PDHCHR          ; SAVE POSSIBLE INPUT CHAR
633                     ;
```

```
634 B894 A47A            LDY   PDHICD          ; GET IOCB DISPL
635 B896 A2FC            LDX   #$FC            ; AND X VALUE FOR 4 BYTES
636 B898 B53A    PDHM2   LDA   $3A,X           ; GET APPLE SWITCH VALUE
637 B89A 990BBD          STA   ICAUX2,Y        ; PUT INTO IOCB
638 B89D 68              PLA                   ; RESTORE SWITCH
639 B89E 953A            STA   $3A,X           ; FROM STACK
640 B8A0 88              DEY
641 B8A1 E8              INX
642 B8A2 D0F4            BNE   PDHM2           ; BR IF MORE TO MOVE
643                ;
644 B8A4 F01F            BEQ   AEDSTA          ; DONE
645                ;
646      B8A6    PDHGO   EQU   *
647 B8A6 A57B            LDA   PDHFLG          ; IF OUTPUT
648 B8A8 F003            BEQ   PDHBO           ; BR
649 B8AA 6C3800          JMP   ($38)
650                ;
651 B8AD A57C    PDHBO   LDA   PDHCHR          ; LOAD DATA
652 B8AF 4980    PDHOSW  EOR   #$80            ; INVERT MSB
653 B8B1 6C3600          JMP   ($36)           ; OUTPUT CHAR
654
```

```
                                PAGE 'APPLE EDITOR DEVICE HANDLER'
                        ;
                        ;       AEDDH - APPLE EDITOR DEVICE HANDLER
                        ;
                        AEDDH
B8B4 C0B8                       DB      @@AEDOPN        ; OPEN
B8B6 C5B8                       DB      @@AEDSTA        ; STATUS
B8B8 CCB8                       DB      @@AEDGBT        ; GET BYTE
B8BA F4B8                       DB      @@AEDPBT        ; PUT BYTE
B8BC C5B8                       DB      @@AEDSTA        ; CLOSE
B8BE C5B8                       DB      @@AEDSTA        ; DEVICE DEPENDENT VECTOR
                        ;
                        AEDOPN
B8C0 A900                       LDA     #0              ;SET OUFLAG=0
B8C2 8DFAB8                     STA     AEDFLG
                        ;
                        AEDINT
                        AEDSTA
B8C5 A001                       LDY     #ICSOK          ; SET OK STATUS
                        !AERTN
B8C7 A57C                       LDA     AEDCHAR         ;GET DATA CHAR
B8C9 4980                       EOR     #$80            ; INVERT MSB
                        AEDDDC
B8CB 60                         RTS                     ;AND RETURN
                        ;
                        AEDGBT
B8CC ACFAB8                     LDY     AEDFLG          ;GET FLAG/COUNT
B8CF D010                       BNE     !AEDG1          ;BR NOT ZERO
                        ;
B8D1 A98D                       LDA     #$8D
B8D3 8533                       STA     $33
B8D5 A200                       LDX     #0
B8D7 2075FD                     JSR     $FD75           ; GET A LINE
B8DA E8                         INX                     ; INC BY 1
B8DB 8EFBB8                     STX     AEDCNT          ;SAVE LINE SIZE
B8DE ACFAB8                     LDY     AEDFLG          ;GET ZERO COUNT
                        ;
                        !AEDG1
B8E1 B90002                     LDA     $200,Y          ;GET DATA CHAR
B8E4 857C                       STA     AEDCHAR         ;SAVE CHAR
B8E6 C8                         INY                     ;INC TO NEXT
B8E7 CCFBB8                     CPY     AEDCNT          ;XFR ALL CHARS YET
B8EA 9002                       BCC     !AEDG2          ;BR IF NOT
B8EC A000                       LDY     #0              ;SET FLAG=0
                        ;
B8EE 8CFAB8             !AEDG2  STY     AEDFLG          ;SET NEW COUNT/FLAG
B8F1 4CC5B8                     JMP     AEDSTA          ;GO GET STATUS & RETURN
                        ;
        B8F4            AEDPBT  EQU     *
B8F4 20AFB8                     JSR     PDHOSW          ; OUTPUT CHAR
B8F7 4CC5B8                     JMP     AEDSTA          ;GO END STATUS
                        ;
        B8FA            AEDFLG  RMB     1               ; EDITOR FLAG
        B8FB            AEDCNT  RMB     1               ; EDITOR COUNT
                        ;
```

OPTIMIZED SYSTEMS SOFTWARE

DISK FILE MANAGER

for the Apple II (R)

Feb 1980

Version 1.0

Optimized Systems Software
Shepardson Microsystems, Inc.
20823 Stevens Creek Blvd, Bldg C4-H
Cupertino, CA 95014
Telephone: 408-257-9900

TABLE OF CONTENTS

INTRODUCTION

The OSS DISK FILE MANAGER runs under the OSS operating system as a Device Handler. The DFM has two entries in the Device Table. The "A:" device is for files located on disk 1 in slot 6. The "B:" device is for files on disk 2 in slot 6. All file manager functions are accessed through the operating system via the IOCBs.

An OSS disk is organized to contain 719 (or 680 if a boot is included) 128 bytes sectors numbered 0 through 719. The file manager reserves 9 sectors for the file management functions. Eight of the reserved sectors are the file directory. Each file directory sector can contain eight file entries; thus, an OSS disk may contain a maximum of 64 files.

FILE NAMES

The DFM accesses files in the file directory via an eleven character file name which the user specifies in the filename portion of the Operating System filespec. A DFM filename as received in the filespec has the general form:

primary-name.extension-name

The primary file name must start with an aplha character (A-Z) and may contain up to seven following aplhanumeric (A-Z,0-9) characters. The extension filename may contain from zero to three aplhanumeric characters. The DFM will pad the primary name to eight characters with blanks. The extension name will be padded to three charecters with blanks.

The DFM filename received in the filespec may also contain the "wild card" search characters "*" and "?". The "?" is interpeted as "any character" in the directory search-for-match operation. A file name of eleven "?" would match with any and all file names during a directory search. The "*" wild card is used to cause a file name to be padded with "?" characters rather than blank characters. The file name "*.*" is a substitute for a file name of eleven "?" characters.

FILE MANAGER FUNCTIONS

The file manager performs the following file management functions:

Open Output	Open a file (new or old) for output at the start of the file.
Open Append	Open a file (old) for output at the end of the file.
Open Update	Open a file (old) for modification of existing records.
Open Directory	Open the directory for output of ASCII formatted file information.
Close	Close and open file.
Get Byte	Get next sequential byte from file open for input, update, or directory.
Put Byte	Put next sequential byte to a file opened for output, append or update.
Note	For purpose of random access, obtain the disk address of the next byte to be used for GET or PUT
Point	Set the disk address of the next byte to GET or PUT. The file must be open for update to do Point.
Erase	Erase a file or files.
Protect	Protect a file or files from modification or erasure.
Unprotect	Unprotect a protected file.
Rename	Rename a file or files.
Status	Obtain the status of a file.

All file manager functions are performed through OS using the system IOCBs (see OS manual). Various applications such as CP/A, BASIC and EASMD provide various levels of automatic access to file management functions.

FUNCTION DETAILS

OPEN OUTPUT

```
IOCB COMMAND            1
IOCB AUX1               8
IOCB BUFADR             Address of filespec
```

The indicated file is open for output from the relative byte zero of the file. If the file already exists and is not protected, the existing file will be ERASEd before opening the named file as a new file. If the file does not exist, it will be created. Wild card characters are used to find the first and only the first match when searching for an existing file. If wild card characters are used and an existing file was not found, the wild card character will be changed to blanks. If an existing file is found, the new file name will be the old file name. A file OPENed for output will not appear in the directory until it has been CLOSEd. If an output file is not properly CLOSEd, some or all of the sectors that were acquired for it may be lost to the system.

OPEN INPUT

```
IOCB COMMAND            1
IOCB AUX1               4
IOCB BUFADR             Address of filespec
```

The indicated file is OPENed for input. Any wild card characters are used to search for the first, and only the first match. If the file is not found, a "FILE NOT FOUND" error will be returned, and no file will be OPENed.

OPEN APPEND

```
IOCB COMMAND            1
IOCB AUX1               5
IOCB BUFADR             Address of filespec
```

The indicated file is OPENed for APPENDing data to the end of the file if the file is not protected. The rules for the file name search are the same as for INPUT. The file must exist. If a file OPENed for APPEND is not properly CLOSEd, the APPENDed data will be lost and the existing file will be unmodified. Non-closure of files OPENed for APPEND may cause some or all of the sectors containing the APPENDed data to be lost to the system.

OPEN UPDATE

```
IOCB COMMAND            1
IOCB AUX1               12  ($0C)
IOCB BUFADR             Address of filespec
```

The indicated file will be OPENed for UPDATE modifications provided it

is not protected. The rules for directory searching are the same as for INPUT. The file must exist. The file I/O pointer is set for the first file data byte. GET and PUT functions are both valid for UPDATE and may be intermixed as desired. If a file OPENed for UPDATE is not properly CLOSEd, a sectors worth of updates may be lost. A file opened for update cannot be extended beyonf its end-of-file.

OPEN DIRECTORY

IOCB COMMAND	1
IOCB AUX1	6
IOCB BUFADR	Address of filespec

The directory is OPENed for input to the caller via GETBYTE. The DFM will format each matched file name into an ASCII line suitable for printing or other processing. The line format is as follows:

CHARACTERS	
0	Protect code, "*" if protected else blank
1	Blank
2 - 9	Primary file name
10 - 12	Extension filename
13	Blank
14 - 16	Count of sectors used by the file
17	Carriage return ($0D)

The last line will contain the number of free sectors available in characters 0 through 2, followed by "FREE SECTORS" and a carriage return. An attempt to get data bytes beyond the last line's carriage return will result in the end-of-file error.

The wild card characters are used in searching the directory. All file name matches that are found will be formatted and returned. If no matches are found, only the free sectors line will be returned. The filespec "*.*" will return all file entries.

CLOSE

IOCB COMMAND	12 ($0C)

The indicated OPEN file is CLOSEd.

GETBYTE

IOCB DATA - see OS documentation

The next sequential data byte is returned (usually to OS) in the A register. The OS provides for data buffering. If an attempt is made to read beyond the end-of-file, the "END-OF-FILE" error will be returned. If the byte read is the last byte before the end-of-file, the end-of-file look ahead condition code will be returned.

PUTBYTE

IOCB information - See OS manual

The data in the (usually OS supplied) A-register will be put in the next sequential file location. If an attempt is made to write beyond the end-of-file in an UPDATE operation, the "END-OF-FILE" error will be returned.

NOTE

IOCB COMMAND	38 ($26)
IOCB AUX3	Sector number (low)
IOCB AUX4	Sector number (high)
IOCB AUX5	Sector byte displacement (zero relative)

Obtain the disk address of the NEXT sequential byte to be accessed. The NOTE and POINT commands are used to build user directories for random or direct access operations.

POINT

IOCB COMMAND	37 ($25)
IOCB AUX3	Sector number (low)
IOCB AUX4	Sector number (high)
IOCB AUX5	Sector byte displacement

Set the disk address of the NEXT byte to be accessed. The file must be OPENed for UPDATE. If the indicated sector does not belong to the file that is OPENed, then an error will be returned. If the sector byte displacement is greater than that sectors current data length, then an error will be returned.

ERASE

IOCB COMMAND	33 ($21)
IOCB BUFADR	Address of filespec

The indicated file or files will be ERASEd unless they are protected. The wild card characters are used to find all matching entries in the directory. Warning: the filespec *.* will ERASE ALL unprotected files.

PROTECT

IOCB COMMAND	35 ($23)
IOCB BUFADR	Address of filespec

The indicated file or files will be protected against change and/or ERASure. The file name search is the same as for ERASE.

UNPROTECT

IOCB COMMAND	36 ($24)
IOCB BUFADR	Address of filespec

The indicated file or files will be UNPROTECTed. The file name search is the same as for ERASE.

RENAME

IOCB COMMAND	32 ($20)
IOCB BUFPTR	Address of filespec

The indicated file or files will be RENAMEd. The filespec contains the name of the files to be searched for under the same rules as ERASE. Following the search argument filespec is the new filename. The two filespecs must be separated by at least one non alphanumeric (A-Z, 0-9) (and non "*" or "?") characters. The new filename must not contain the device name "X:" part of a filespec. The new filename may contain wild card characters. Any wild card character in the new filename will be replaced by the corresponding character in the old filename. A file that is PROTECTed will not be RENAMEd.

STATUS

IOCB COMMAD	13 ($0D)
IOCB BUFPTR	Address of filespec

The STATUS of the indicated file is returned. The wild card characters are used in the directory search. The first file found, and only the first file found will be STATUSed. The STATUS will indicate if the file exists and, if it does, whether it is PROTECTed or not. The $01 (normal) status indicates the file exists, for other status values see Return Code section.

RETURN CODES

The following codes are returned by the File Managerin the IOCB status byte and in the Y register.

CODE		MEANING
$01		Normal operation ending.
$03		End-of-file look ahead. The byte just returned is the last byte in file.
$81	(129)	No disk in drive, or device error.
$83	(131)	Data I/O error.
$87	(135)	Disk write protected.
$A1	(161)	All sectors buffers in use
$A2	(162)	Disk full. No free sectors
$A3	(163)	I/O error reading system sector (directory or bit map)
$A4	(164)	Attempted to read a sector that was not part of currently OPENed file.
$A5	(165)	Invalid file name
$A6	(166)	Point information in error
$A7	(167)	File protected.
$A8	(168)	Invalid DFM command.
$A9	(169)	Directory full. Contains 64 files.
$AA	(170)	File not found in directory.
$AB	(171)	Point command issued when file was not OPEN for UPDATE.

DISK I/O

The OSS disk that has been formatted without a boot record contains 720 sectors of 128 bytes each. The sectors are numbered 0 through 719 (decimal). The routine, DIOENT ($AB00), performs the actual reading and writing of the sectors using the Disk I/O Block (DIOB) at $BD80. The DIOENT routine is normally used only by the DFM; however, it is easily accessed by user programs. The reading or writing of disk sectors requires only that the correct information be placed in the DIOB and a subroutine call made to DIOENT.

DIOB DETAILS

LOCATION	FIELD	USAGE
$BD80	DRIVE	Disk drive to use. 1 = Slot 6, Drive 1 (A:) 2 = Slot 6, Drive 2 (B:)
$BD81	COMMAND	Command function. 1 = Read sector 2 = Write sector
$BD82	STATUS	I/O Status. $01 = Normal $81 = Device Error $83 = Data Error $84 = Invalid Command $87 = Write Protect
$BD83	BUFFER ADDRESS	Address of 128 byte buffer for data I/O. (Low, High order)
$BD85	SECTOR NUMBER	Absolute Sector Number. (0-$2CF). (Low, High order).

DFM BUFFERS

The Disk File Manager requires 256 bytes of system buffer space plus one 128 byte buffer for each concurrently opened file. The system as delivered provides for a 768 byte buffer space at $A800 (to $AB00). The 768 bytes will provide for four (4) concurrently opened files. The user may change both the address space used for the buffers and the number of sector buffers used. Location $ADA2 (DFMBUF in OS system memory map) contains the start address of the buffer space. Location $ADA1 (DFMDIR) contains an allocation direction switch. If the direction switch is $80, the buffer space will be allocated from the start address toward location $0000. If the direction switch is $00, the buffer space will be allocated from the start address toward location $FFFF. Location $ADA0 (DFMNUMF) contains the number of sector buffers to allocate. The space required for an allocation is (256 + number buffers * 128).

Most OSS system programs are designed to end at location $A800. if the number of buffers is increased beyond the four provided for, the buffer space should be moved ($B00 up is suggested). If less than 4 buffers are required, the space from $A800 to $A980 may be reclaimed for user application ram (in 128 byte chunks).

The buffer space parameters may be permanently changed if INIT is used to re-write the boot.