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

DSS Control Program/Apple is Copyright (c) 1980
Shepardson Microsystems, Inc.

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

Apple II and Disk II are registered trademarks of Apple Computer, Inc.
# TABLE OF CONTENTS

**GENERAL INFORMATION** ............................................................................................................. 1
  Intrinsic Commands ......................................................................................................................... 1
  Extrinsic Commands ......................................................................................................................... 1

**DEFAULT DEVICE** ............................................................................................................................ 2

**COMMAND DETAILS** ..................................................................................................................... 3
  Save ................................................................................................................................................ 3
  Load .............................................................................................................................................. 3
  Run ............................................................................................................................................... 3
  Directory .................................................................................................................................... 4
  Erase ........................................................................................................................................... 4
  Protect ....................................................................................................................................... 4
  Unprotect .................................................................................................................................... 4
  Rename ....................................................................................................................................... 4
  Init .............................................................................................................................................. 5
  Dupdsk ......................................................................................................................................... 6
  Copy ............................................................................................................................................ 7

**USER WRITTEN EXTRINSIC COMMANDS** .................................................................................... 8
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECTORY</td>
<td>List directory</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save a program</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load a program</td>
</tr>
<tr>
<td>RUN</td>
<td>Run a program already in RAM</td>
</tr>
<tr>
<td>ERASE</td>
<td>Erase a file from it’s medium or change</td>
</tr>
<tr>
<td>PROTECT</td>
<td>Protect a file from erasure</td>
</tr>
<tr>
<td>UNPROTECT</td>
<td>Unprotect a protected file</td>
</tr>
<tr>
<td>RENAME</td>
<td>Rename a file</td>
</tr>
</tbody>
</table>

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 description of the command's operation.

SAVE

SAVE[E] filespec start-hex-adr end-hex-adr
SAVE TEST 800 800

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

LOAD[D] filespec
LOAD TEST

The specified file will be loaded. The file's 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

DIRECTORY optional-fil secs
DIR *.COM

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

    default-device: *.

    filenames will be used.

ERASE

ERASE filenames
ERASE TEST.*

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

PROTECT

PROTECT filenames
PRO BASIC.COM

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

UNPROTECT

UNPROTECT filenames
UNP DATA.TST

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

RENAME

RENAME old-fil secs new-fil secs
REN QLOP ACTS.NEW

The files matching the old-fil secs are renamed according to the new-fil secs.
INIT (Extrinsic command)

INIT (no parameters)

The INIT command is used to physically and logically initialize 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 DOS system. CP/A will jump to the extrinsic command with IOCBO numbers 1 through 7 closed. IOCBO number 0 is open for the current console device for both input and output. IOCBO 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

OSS Operating System is Copyright (c) 1980
Shepardson Microsystems, Inc.

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

Apple II and Disk II are registered trademarks of Apple Computer, Inc.
# TABLE OF CONTENTS

**GENERAL**
- INFORMATION ........................................... 1
  - Introduction to IOCBS ................................. 1
  - Filespec Description ................................ 1

**IOCBs** ................................................. 2
- IOCBs Fields ............................................. 2
- IOCB Field Names ....................................... 2
- IOCB Description ....................................... 2

**OS COMMANDS** ......................................... 3
- The Three Types of Commands ........................... 3
  - Open .................................................... 3
  - Getrecord ............................................. 4
  - Getline ................................................ 4
  - Putrecord ............................................. 4
  - Putline ................................................ 4
  - Close .................................................. 4
  - Status ............................................... 4
  - Device Dependent .................................... 4

**OS STATUS** ............................................... 5
- Returned Status And Error Codes ....................... 5

**USING THE OS FROM ASSEMBLY LANGUAGE** ................. 5
- Calling Conventions .................................... 5

**DEVICE HANDLERS** ..................................... 6
- Device Handler Table ................................... 6
- Device Handler Vector Table ............................ 6
- Device Handler Coding Conventions .................... 6

**DEVICE E** ................................................ 7
- Console I/O ............................................. 7
- IOCB #0 ................................................ 7

**DEVICE Pm** ................................................ 8
- Using Apple Ports With OSS ............................. 8

**SYSTEM MEMORY MAP** .................................. 9
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 interprets 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 identifies 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.
There are Eight IOCBS in the system. The first IOC (IOCB #0) is located at address IOC (see memory map). Each IOC is 16 bytes in length and all IOCBS are contiguous. The following details the specific use of each IOC byte.

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

The OS commands fall into three general classes.

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

The specified IODCB 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 IODCB. The device handler open routine will be called to provide whatever device open functions are required. Once the IODCB 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 IODCB 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 interprets all commands above a certain value to be Device Dependent Commands. If the IODCB 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 IODCB is "open" for the one command only).

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

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>VALUE (HEX)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>$01</td>
<td>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.</td>
</tr>
<tr>
<td>GETRECORD</td>
<td>$04</td>
<td>A record of length BUFLEN will be moved into the buffer pointed to by BUFADR. The IODCB must have been OPENed for input.</td>
</tr>
<tr>
<td>Command</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>GETLINE</td>
<td>$05</td>
<td>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 IOCBB must have been OPENed for input.</td>
</tr>
<tr>
<td>PUTRECORD</td>
<td>$08</td>
<td>A record of length BUFLEN will be sent to the device from the buffer pointed to by BUFADR. The IOCBB must have been OPENed for output.</td>
</tr>
<tr>
<td>PUTLINE</td>
<td>$09</td>
<td>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 IOCBB must have been OPENed for output.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>$0C</td>
<td>The IOCBB and file are closed.</td>
</tr>
<tr>
<td>STATUS</td>
<td>$0D</td>
<td>The device will return a status byte in the IOSTAT field. The status returned is Device Dependent. The IOCBB need not have been OPENed. If not OPEN, BUFADR must point to a file specification.</td>
</tr>
<tr>
<td>DEVICE DEPENDENT</td>
<td>$0E-$7F</td>
<td>The DEVICE DEPENDENT commands are sent directly to the device handler. The IOCBB 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.)</td>
</tr>
</tbody>
</table>
All OS operations return a status value in the IOSTAT field. OS conventions is that status values of $80 or greater indicate some sort of error.

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

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:

```assembly
LDX #50
JSR OS
TYA
BMI ERROR
BPL GOODIO
```

; USING IOCB #5
; CALL OS
; SET PROCESSOR STATUS FLAG
; BRANCH IF ERROR
; 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 available to the user. The format of a Device Handler table entry is as follows:

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

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 IOC B number (0-7) times 16. The user may use the register to directly access the specified IOC B 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 DHZPQE (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 initialization. All OSS system programs (CP/A, BASIC, DMGR, etc) use IOC B #0 for console I/O. No OSS system routine closes IOC B #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 IOC B #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 IOCDB. When a Pn: data byte I/O is called for, the following sequence occurs:

1) The device address saved in the IOCDB are swaped 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 the Pn: device handler.

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

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.
<table>
<thead>
<tr>
<th>LOCATION</th>
<th>LABEL</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFFD</td>
<td>CPRTN</td>
<td>JMP CP/A</td>
</tr>
<tr>
<td>BFFA</td>
<td>SINIT</td>
<td>JMP system initializer</td>
</tr>
<tr>
<td>BFF8</td>
<td>HIMEM</td>
<td>HIMEM</td>
</tr>
<tr>
<td>BFF6</td>
<td>LOMEM</td>
<td>LOMEM</td>
</tr>
<tr>
<td>BFF5</td>
<td>OSVER</td>
<td>OSS version number</td>
</tr>
<tr>
<td>BDA0</td>
<td>OSENT</td>
<td>OS entry address</td>
</tr>
<tr>
<td>BDB7</td>
<td>DHTAB</td>
<td>Device table (8 devices)</td>
</tr>
<tr>
<td>BDB0</td>
<td>DIOB</td>
<td>Disk I/O Block</td>
</tr>
<tr>
<td>BD00</td>
<td>IOC8</td>
<td>IOC8s (8 IOC8s)</td>
</tr>
<tr>
<td>BCFE</td>
<td>DEFDIV</td>
<td>Default Drive (ASCII character)</td>
</tr>
<tr>
<td>B900</td>
<td>CPAENT</td>
<td>CP/A entry address</td>
</tr>
<tr>
<td>BB50</td>
<td></td>
<td>E: and P: device handlers</td>
</tr>
<tr>
<td>ADA0</td>
<td>DFMNMF</td>
<td>Number of file buffers (4 default)</td>
</tr>
<tr>
<td>ADA1</td>
<td>DFMDIR</td>
<td>File buffer allocation direction ($80)</td>
</tr>
<tr>
<td>ADA2</td>
<td>DFMBUF</td>
<td>File buffers start address ($A000)</td>
</tr>
<tr>
<td>AB00</td>
<td>DIDENT</td>
<td>Disk I/O Routine</td>
</tr>
<tr>
<td>AB00</td>
<td></td>
<td>File buffers</td>
</tr>
<tr>
<td>0B00</td>
<td></td>
<td>User Ram</td>
</tr>
<tr>
<td>0400</td>
<td></td>
<td>Apple screen buffer</td>
</tr>
<tr>
<td>03F9</td>
<td></td>
<td>JMP go-location</td>
</tr>
<tr>
<td>03F0</td>
<td></td>
<td>Auto start Rom vectors</td>
</tr>
<tr>
<td>0280</td>
<td>CMDLINE</td>
<td>CP/A command line</td>
</tr>
<tr>
<td>0200</td>
<td></td>
<td>Line buffer and work space</td>
</tr>
<tr>
<td>0100</td>
<td></td>
<td>6502 stack</td>
</tr>
<tr>
<td>0080</td>
<td></td>
<td>Application zero page Ram</td>
</tr>
<tr>
<td>007F</td>
<td>DHZPGE</td>
<td>Top of Device Handlers temps</td>
</tr>
<tr>
<td>0079</td>
<td>DHZPG</td>
<td>Start of Device Handlers temps</td>
</tr>
<tr>
<td>0068</td>
<td></td>
<td>OSS system zero page</td>
</tr>
<tr>
<td>0050</td>
<td></td>
<td>Available zero page</td>
</tr>
<tr>
<td>0020</td>
<td></td>
<td>Apple Monitor Ram</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>Available zero page</td>
</tr>
</tbody>
</table>
PORT DEVICE HANDLER

580  APPLE PORT DEVICE

583  PDHCHR EQU AEDCHAR ; DATA CHAR
584  PDHICD EQU DH2P+1 ; IOC8B DISPL
585  PDHFLG EQU DH2P+2 ; I/O FLAG
586  ORG $B50
587  B50 PORDTH EQU *
589  B50 5CBB DB @PDHOPN ; OPEN
590  B52 C5BB DB @AEDSTA ; STATUS
591  B54 79BB DB @PDHGBT ; GET BYTE
592  B56 73BB DB @PDHBPT ; PUT BYTE
593  B58 C5BB DB @AEDSTA ; CLOSE
594  B5A C5BB DB @AEDSTA ; DEVICE DEPENDENT
595  B5C PDHOPN EQU * ; OPEN PORT N
596  B5D A900 LDA #0 ; BET ZERO TO
598  B5E 9D0FD STA ICAUX4,X ; LOW ADDR BYTE OUT
599  B61 9D0DB STA ICAUX4,X ; LOW ADDR BYTE IN
600  B64 BD01BD LDA ICN00,X ; GET DEVICE NO
602  B67 2907 AND #$07 ; ISOLATE 3 LSB
603  B69 09C0 ORA #$C0 ; OR IN ADR HI
604  B6B 9D0DB STA ICAUX3,X ; HIGH ADDR BYTE OUT
605  B6C 9D0CBD STA ICAUX3,X ; HIGH ADDR BYTE IN
606  B71 D052 BNE AEDSTA ; DONE
607  B73 PDHOPB EQU * ; PUT DATA BYTE
609  B75 A900 LDA #0 ; SAVE DATA BYTE
610  B77 F002 BEQ PDHOP ; INDICATE OUTPUT
612  B79 PDHBRT EQU *
614  B79 A9FF LDA #$FF ; INDICATE INPUT
615  B7B PDHCP EQU *
616  B7D 657B STA PDHFLG ; SAVE FLAG
618  B7D BA TXA
619  B7E AB TAY ; IOC8B DISPL TO Y
620  B7F A203 LDX #3 ; SAVE X FOR 4 BYTE MOVE
621  B81 B536 PDHM1 LDA #36,X ; GET APPLE SWITCH BYTE
623  B83 4B PHA ; SAVE ON STACK
624  B84 9D0CBD LDA ICAUX3,Y ; MOVE VECTOR FROM IOC8B
625  B87 9536 STA #36,X ; TO APPLE SWITCH BYTE
626  B89 CB INY
627  B8A CA DEX
628  B8B 10F4 BPL PDHM1 ; BR IF MORE TO MOVE
629  B8D B47A STY PDHICD ; SAVE IOC8B DISPL
631  B8F 20A6B8 JSR PDHCP ; GO DO I/O
632  B92 B57C STA PDHCHR ; SAVE POSSIBLE INPUT CHAR
633
634 BB94 A47A LDV PDHICD ; GET IOCBL DISPL
635 BB96 A2FC LDX **FC ; AND X VALUE FOR 4 BYTES
636 BB9B B53A PDHM2 LDA $3A, X ; GET APPLE SWITCH VALUE
637 BB9A 990BBD STA ICAUX2, Y ; PUT INTO IOCBL
638 BB9D 68 PLA ; RESTORE SWITCH
639 BB9E 953A STA $3A, X ; FROM STACK
640 BBA0 BB DEY
641 BBA1 E8 INX
642 BBA2 D0F4 BNE PDHM2 ; BR IF MORE TO MOVE
643
644 BBA4 F01F BEQ AEDSTA ; DONE
645
646 BBA6 PDHGO EQU *
647 BBA6 A57B LDA PDHFLG ; IF OUTPUT
648 BBAB F003 BEQ PDHDO ; BR
649 BBAA 6C3B00 JMP ($38)
650
651 BBAD A57C PDHDO LDA PDHCHR ; LOAD DATA
652 BBAE 49B0 PDHOSW EOR $80 ; INVERT MSB
653 BBF1 6C3600 JMP ($36) ; OUTPUT CHAR
AEDDH - APPLE EDITOR DEVICE HANDLER

655
656
657
658
659 BB4 C0BB DB @@AEDOPN ; OPEN
660 BB6 C5BB DB @@AEDSTA ; STATUS
661 BB8 CCBB DB @@AEDGBT ; GET BYTE
662 BBAA F4BB DB @@AEDPBT ; PUT BYTE
663 BBBC C5BB DB @@AEDSTA ; CLOSE
664 BBEE C5BB DB @@AEDSTA ; DEVICE DEPENDENT VECTOR
665
666
667 BBCC A900 LDA #0 ; SET OUTFLAG=0
668 BBC2 BDFABB STA AEDFLG
669
670
671
672 BBC5 A001 LDY #$C0 ; SET OK STATUS
673
674 BBCC A57C LDA AEDCHAR ; GET DATA CHAR
675 BBCC 49B0 EOR #$80 ; INVERT MSB
676
677 BBCC 60 RTS ; AND RETURN
678
679
680 BBCC ACFABB LDY AEDFLG ; GET FLAG/COUNT
681 BBCC D010 BNE !AEDG1 ; BR NOT ZERO
682
683 BBDD A9BD LDA #$8D
684 BBDD B533 STA #$33
685 BBDD A200 LDX #0
686 BBDD 2075FD JBR #$FD75 ; GET A LINE
687 BBDA EB INX ; INC BY 1
688 BBDE 8FBB8 STY AEDCNT ; SAVE-LINE-SIZE
689 BBDE ACFABB LDY AEDCNT ; GET ZERO COUNT
690
691
692 BBE1 B90002 LDA #200,Y ; GET DATA CHAR
693 BBE1 B57C STA AEDCHAR ; SAVE CHAR
694 BBE6 CC INV ; INC TO NEXT
695 BBE7 CCFBB8 CPY AEDCNT ; XFR ALL CHARs YET
696 BBEA 9002 BCC !AEDG2 ; BR IF NOT
697 BBEC A000 LDY #0 ; SET FLAG=0
698
699 BBEE BCFABB !AEDG2 STY AEDFLG ; SET NEW COUNT/FLAG
700 BBF1 4CC5BB JMP AEDSTA ; GO SET STATUS & RETURN
701
702 BBF4 AEDPBT EQU *
703 BBF4 20AFFB JBR PDHEW ; OUTPUT CHAR
704 BBF7 4CC5BB JMP AEDSTA ; GO END STATUS
705
706 BBFA AEDFLG RMB 1 ; EDITOR FLAG
707 BBFB AEDCNT RMB 1 ; EDITOR COUNT
OPTIMIZED SYSTEMS SOFTWARE

DISK FILE MANAGER

for the Apple II (R)

Feb 1980

Version 1.0

OSS Disk File Manager is Copyright (c) 1980
Shepardson Microsystems, Inc.

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

Apple II and Disk II are registered trademarks of Apple Computer, Inc.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Drive A: and B:</td>
<td>1</td>
</tr>
<tr>
<td>Disk Organization</td>
<td>1</td>
</tr>
<tr>
<td>FILE NAMES</td>
<td>2</td>
</tr>
<tr>
<td>Primary and Extension Names</td>
<td>2</td>
</tr>
<tr>
<td>Wild Card Search Characters</td>
<td></td>
</tr>
<tr>
<td>FILE MANAGER FUNCTION</td>
<td>3</td>
</tr>
<tr>
<td>FUNCTION DETAILS</td>
<td>4</td>
</tr>
<tr>
<td>Open Output</td>
<td>4</td>
</tr>
<tr>
<td>Open Input</td>
<td>4</td>
</tr>
<tr>
<td>Open Update</td>
<td>4</td>
</tr>
<tr>
<td>Open Directory</td>
<td>6</td>
</tr>
<tr>
<td>Close</td>
<td>6</td>
</tr>
<tr>
<td>Getbyte</td>
<td>7</td>
</tr>
<tr>
<td>Putbyte</td>
<td>7</td>
</tr>
<tr>
<td>Note</td>
<td>7</td>
</tr>
<tr>
<td>Point</td>
<td>7</td>
</tr>
<tr>
<td>Erase</td>
<td>8</td>
</tr>
<tr>
<td>Protect</td>
<td>8</td>
</tr>
<tr>
<td>Unprotect</td>
<td>8</td>
</tr>
<tr>
<td>Rename</td>
<td>9</td>
</tr>
<tr>
<td>Status</td>
<td>9</td>
</tr>
<tr>
<td>RETURN CODES</td>
<td>10</td>
</tr>
<tr>
<td>DISK I/O</td>
<td>11</td>
</tr>
<tr>
<td>DIOB DETAILS</td>
<td>11</td>
</tr>
<tr>
<td>DFM BUFFERS</td>
<td>12</td>
</tr>
</tbody>
</table>
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 I/O CBs.

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 character with blanks.

The DFM filename received in the filespec may also contain the "wild card" search characters "*" and "?". The "?" is interpreted 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:

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

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

<table>
<thead>
<tr>
<th>IOCБ COMMAND</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCБ AUX1</td>
<td>8</td>
</tr>
<tr>
<td>IOCБ BUFADR</td>
<td>Address of filespec</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IOCБ COMMAND</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCБ AUX1</td>
<td>4</td>
</tr>
<tr>
<td>IOCБ BUFADR</td>
<td>Address of filespec</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IOCБ COMMAND</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCБ AUX1</td>
<td>5</td>
</tr>
<tr>
<td>IOCБ BUFADR</td>
<td>Address of filespec</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IOCБ COMMAND</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCБ AUX1</td>
<td>12 ($OC)</td>
</tr>
<tr>
<td>IOCБ BUFADR</td>
<td>Address of filespec</td>
</tr>
</tbody>
</table>

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 beyond its end-of-file.
OPEN DIRECTORY

IDCB COMMAND  1
IDCB AUX1  6
IDCB BUFADR  Address of filespec

The directory is OPENed for input to the caller via QETBYTE. 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

IDCB COMMAND  12 ($OC)

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

<table>
<thead>
<tr>
<th>IOCB COMMAND</th>
<th>38 ($26)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCB AUX3</td>
<td>Sector number (low)</td>
</tr>
<tr>
<td>IOCB AUX4</td>
<td>Sector number (high)</td>
</tr>
<tr>
<td>IOCB AUX5</td>
<td>Sector byte displacement (zero relative)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IOCB COMMAND</th>
<th>37 ($25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCB AUX3</td>
<td>Sector number (low)</td>
</tr>
<tr>
<td>IOCB AUX4</td>
<td>Sector number (high)</td>
</tr>
<tr>
<td>IOCB AUX5</td>
<td>Sector byte displacement</td>
</tr>
</tbody>
</table>

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

IDCB COMMAND  33  ($21)
IDCB 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

IDCB COMMAND  35  ($23)
IDCB 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

IDCB COMMAND  36  ($24)
IDCB 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 COMMAND  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 Manager in the IOCB status byte and in the Y register.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>$01</td>
<td>Normal operation ending.</td>
</tr>
<tr>
<td>$03</td>
<td>End-of-file look ahead. The byte just returned is the last byte in file.</td>
</tr>
<tr>
<td>$81</td>
<td>(129) No disk in drive, or device error.</td>
</tr>
<tr>
<td>$83</td>
<td>(131) Data I/O error.</td>
</tr>
<tr>
<td>$87</td>
<td>(135) Disk write protected.</td>
</tr>
<tr>
<td>$A1</td>
<td>(161) All sectors buffers in use</td>
</tr>
<tr>
<td>$A2</td>
<td>(162) Disk full. No free sectors</td>
</tr>
<tr>
<td>$A3</td>
<td>(163) I/O error reading system sector (directory or bit map)</td>
</tr>
<tr>
<td>$A4</td>
<td>(164) Attempted to read a sector that was not part of currently OPENed file.</td>
</tr>
<tr>
<td>$A5</td>
<td>(165) Invalid file name</td>
</tr>
<tr>
<td>$A6</td>
<td>(166) Point information in error</td>
</tr>
<tr>
<td>$A7</td>
<td>(167) File protected.</td>
</tr>
<tr>
<td>$A8</td>
<td>(168) Invalid DFM command.</td>
</tr>
<tr>
<td>$AA</td>
<td>(170) File not found in directory.</td>
</tr>
<tr>
<td>$AB</td>
<td>(171) Point command issued when file was not OPEN for UPDATE.</td>
</tr>
</tbody>
</table>
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, DIDENT ($A000), performs the actual reading and writing of the sectors using the Disk I/O Block (DIOB) at $BDB0. The DIDENT 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 DIDENT.

DIOB DETAILS

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>FIELD</th>
<th>USAGE</th>
</tr>
</thead>
</table>
| $BDB0    | DRIVE       | Disk drive to use.  
             | 1 = Slot 6, Drive 1 (A:)  
             | 2 = Slot 6, Drive 2 (B:)  |
| $BDB1    | COMMAND     | Command function.  
             | 1 = Read sector  
             | 2 = Write sector |
| $BDB2    | STATUS      | I/O Status.  
             | $01 = Normal  
             | $81 = Device Error  
             | $83 = Data Error  
             | $84 = Invalid Command  
             | $87 = Write Protect |
| $BDB3    | BUFFER ADDRESS | Address of 128 byte buffer  
             | for data I/O. (Low, High order) |
| $BDB5    | 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 $ADAO (DFMNUMF) contains the number of sector buffers to allocate. The space required for an allocation is (256 + number buffers * 128).

Most DSS system programs are designed to end at location $AB00. If the number of buffers is increased beyond the four provided for, the buffer space should be moved ($800 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.