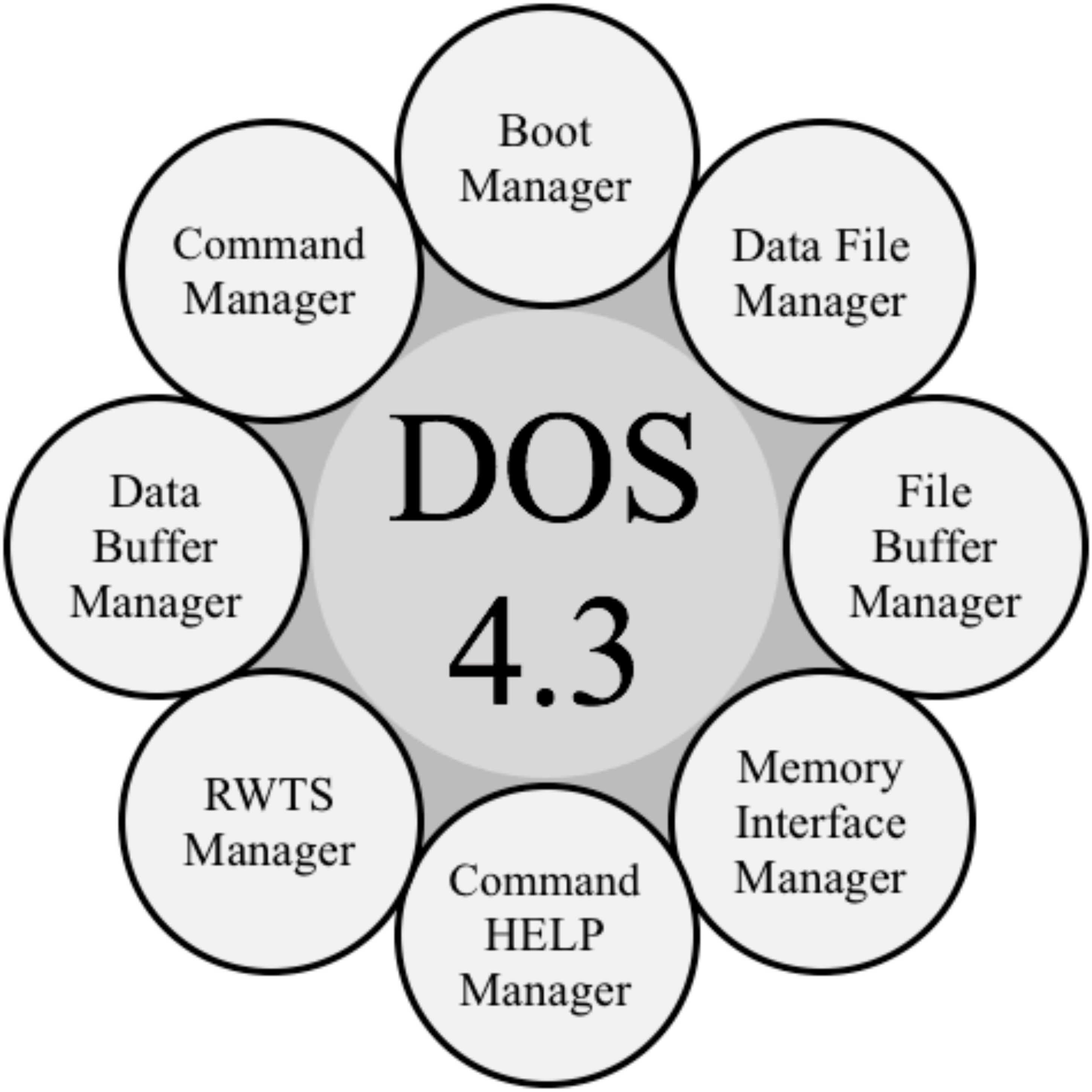
DOS 4.3 File Management System



Walland Philip Vrbancic, Jr.

DOS 4.3

File Management System

Walland Philip Vrbancic, Jr.

lulu

DOS 4.3 File Management System
and this Publication are the
Confidential and Proprietary Intellectual Properties
of
Walland Philip Vrbancic, Jr.

ISBN 978-0-578-71547-6

I am ever so proud to dedicate this Book on the
DOS 4.3 File Management System
and all my previous achievements
to my Parents Wally and Melba
who continuously nourished my intellectual curiosity.

I am ever so grateful to my partner
Carlton D. Wong
who delightfully pretends to understand
what the Hell I am talking about!

If I have seen further than others it is because I have stood on the shoulders of giants.
~~~ Isaac Newton ~~~
~~~

Disclaimer of All Liability

Do not use the DOS 4.3 File Management System or this Book for any mission-critical applications or for any purpose in which a software error or a software failure could cause you financial or material loss. The DOS 4.3 File Management System and this Book are designed to enhance your Apple][computing experience, but they may contain design flaws that could inhibit the proper operation of your computer or result in the loss of recorded data on any storage device connected to your computer. When using the DOS 4.3 File Management System or this Book, you assume all risks associated with the operation of your computer and the potential loss of your data. If these terms are not acceptable to you, please do not use the DOS 4.3 File Management System or this Book.

Walland Philip Vrbancic, Jr., the administrator of applecored.net, makes no warranties either expressed or implied with respect to the DOS 4.3 File Management System or with respect to this Book, its quality, performance, or fitness for any particular purpose. Any risk of incidental or consequential damages resulting from the use of the DOS 4.3 File Management System or the use of information contained in this Book shall be assumed by you, the User. In no event will Walland Philip Vrbancic, Jr., or applecored.net be liable for any direct, indirect, incidental, or consequential damages resulting from any defect or deficiency in the DOS 4.3 File Management System or in this Book.

While all possible attempts have been made to ensure that the information contained within this publication is complete and accurate, the author shall have no liability or responsibility for any errors or omissions, or for any damages or data loss resulting from the use of the information, circuit diagrams, and example software programs contained herein. The author reserves the right to make any changes and/or any improvements to the DOS 4.3 File Management System or to the contents of this publication at any time and without any prior notice.

Preface

When Brian Wiser and Bill Martens discovered my DOS 4.1 documentation and software at applecored.net, they immediately contacted me and wanted Apple Pugetsound Program Library Exchange (A.P.P.L.E.) to publish my DOS 4.1 Manual. Ha! If only this would have happened back in 1982. That's when my co-worker, Randy at Rockwell, and I were actively reading many publications on Apple software and hardware, and Call-A.P.P.L.E. was one of our favorite publications. Needless to say, to be published by any of those computer journals at that time would have been crazy exciting, and certainly a cherished memory for a lifetime. I actually was very close to finishing DOS 4.1 when I agreed to have Call-A.P.P.L.E. publish the DOS 4.1 Manual, Build 45, and provide demo diskette images for DOS 4.1L and DOS 4.1H.

I wanted both versions of DOS 4.1 to provide the user with virtually the same computing experience, albeit the `HELP` command is found only in DOS 4.1H. This desire proved to be somewhat troublesome in that I was limited in memory for DOS 4.1L and I had ample memory for DOS 4.1H. It was the unused memory in DOS 4.1H that was the impetus to introduce the `HELP` command in the first place. At the onset I warned both Wiser and Martens that I could not stop creating more functionality in DOS 4.1, but they were rather insistent on printing the DOS 4.1 Manual for the Apple][community as it was. I finished DOS 4.1 with Build 46. Only Build 46 can now be found at applecored.net as well as its respective PDF.

My next area of exploration for Apple DOS was an attempt to port DOS 4.1H to Auxiliary memory. I was absolutely successful, I might add, but I could not successfully design an interface between *Lisa* (my most favorite 65C02 assembler) and this DOS residing in Auxiliary memory. Over the course of several months in effort, I could not realize a viable solution that would be elegant, save memory, and provide the roadmap for interfacing other utilities and tools to this DOS. But this effort was certainly not wasted! I documented what I had learned about Main and Auxiliary memory management and moved forward to other areas of exploration.

I decided that I would use DOS 4.1H as my initial model for DOS 4.3. Yes, DOS 4.3 does retain the "H" designation for High memory. However, there is no DOS 4.3L. So, I simply refer to my new DOS as DOS 4.3. The question then became, can *Lisa* be ported and function in Auxiliary memory? The answer to that question turns out to be a resounding "Yes!" With DOS 4.3 in Main Language Card memory and *Lisa* in Auxiliary Language Card memory, the user has access to virtually all of Main memory below `0xBE00` for source code, object code, and the symbol list. I saw this configuration simply as an exercise of many potential and new possibilities.

Now, if I can relocate *Lisa* to Auxiliary memory, what about doing the same thing for *Big Mac*? I have to say that this challenge was a bit uneventful because relocating *Big Mac* to Auxiliary Language Card memory was even easier to accomplish. My main focus in *Big Mac* was to align *Sourceror* and *Big Mac* in terms of their *SWEET16* sourcing and assembling abilities, though I do not believe *Big Mac* has ever been able to assemble all of its own *SWEET16* opcodes. This task turned out to be an extraordinary undertaking: I wanted *Sourceror* and *Big Mac* to disassemble/assemble MY version of the *SWEET16* opcodes. I discovered that *Big Mac* could not even assemble its own unique *SWEET16* `EVAL` opcode. This tells me that Bredon probably did not even use *Big Mac* to assemble his own *Big Mac* source code. I have to confess that there still remain two *SWEET16* branches in my disassembled *Big Mac* source code that are wrong, and I do not know their solution to this day. They occur at memory addresses `0xD2C1` and `0xD2DF`. Furthermore, I have yet to discover how to force their execution in order to analyze the resulting behavior in *Big Mac*. I suspect these particular instructions may be part of `MACRO` handling, something I have had no reason to use. *Big Mac* and DOS 4.3 now complement each other beautifully.

During my journey in developing the DOS 4.3 File Management System, I discovered many more layers of File Manager functionality that were almost coded correctly. It beleaguered me no end when I would issue a `CLOSE` statement on the Apple command line in DOS 3.3 or in DOS 4.1, and something would be flushed to the volume in focus. What was it? Why? I found even more examples of questionable logic, wrong logic, and desperate logic. I literally tore apart many of those “weird” routines used by the `CLOSE` statement so that now DOS commands will finish completely and data will be properly flushed. All these issues and many, many more have been resolved in DOS 4.3. The final frontier I tore apart was the `RWTS` manager and format algorithm. Using a utility of my own design to scan a track for its raw data and display the structure of that raw data allowed me to develop my own algorithm for a complete and revolutionary `RWTS` manager.

I know the user will discover many fascinating developments in the DOS 4.3 File Management System: he will be left wondering how he accomplished anything in a timely fashion without having had those developments in any other version of a previous Apple Disk Operating System. I would take that as my greatest compliment.

Enjoy the ride!

Table of Contents

IV. DOS 4.3 Assembly Language Routines 159

V. DOS 4.3 Operational Environment 177

List of Figures

List of Tables

I. Designing Another New DOS

This publication describes the process and products I created when I decided to design and program an enhanced Disk Operating System (DOS) for my Apple //e. Wherever I am able, I have included schematic diagrams, code samples, equations, figures, tables, and representative screen shots to help explain what I have created and the reasons why I did so. As in my previous design of an Apple][DOS, i.e. DOS 4.1L and DOS 4.1H, this has been an incredible journey for me. With DOS 4.3 I have again re-imagined that time when I mostly lived, breathed, and worked on Apple][computers, hardware, and software development continuously for a good period of my life many, many years ago.

1. Introduction

I have been an avid Apple][computer enthusiast, hobbyist, and professional software programmer since 1983 when I became the proud owner of an Apple][+ computer. Besides the Apple][+, my initial system included an Apple][Language Card, a Disk][with an Apple][Disk Controller slot card, an Amdek color monitor, and an Epson MX100 printer with a Grappler+ Printer Interface slot card. During those early years I designed and built my own Apple][peripheral slot cards, made electrical and hardware modifications to my Apple][+ motherboard and keyboard, and wrote a substantial number of software programs using Applesoft BASIC (Applesoft hereafter) and 6502 assembly language. I soon acquired a Videx UltraTerm video display slot card and a Microsoft Z80 slot card. With the Z80 card I began writing Fortran programs that analyzed tomographic reconstructions of the human spinal column. A year or so later I added the Southern California Research Group quikLoader and PROmGRAMER slot cards, a Johnathon Freeman Designs (JFD) Parallel Printer Buffer, and an Axlon RAM Disk 320 and its interface slot card to my system.

I used C language in my professional programming career for the software development of ultra-high-speed data collection systems for tactical radar and sensor development. Now that I am retired from the aerospace industry I have always wanted to dig into, tear apart, and learn the intricacies of the last available DOS for the Apple][+. That DOS, DOS 3.3, was published on August 25, 1980. Then I recently came across another version of DOS 3.3 published years later on January 1, 1983. That DOS contains even more patches for the DOS `APPEND` command and a patch for Apple //e initialization. What I learned from the 1980 publication flabbergasted me: the software is exciting in its originality and concept vis-à-vis it was released just after the publication of Integer BASIC. However, I found the software to be somewhat juvenile in structure and implementation. Apparently, very little attention was given to software design and review. It appeared to me Apple made a strong push to release “something or anything” to consumers and vendors in order to begin marketing software products on diskettes. And history does reveal that Apple Computer did outsource DOS and contracted for it to be delivered within 35 days for $13,000 in April, 1978. Paul Laughton at Shepardson Microsystems wrote Apple’s initial disk operating system using Hollerith cards, a card reader, and a minicomputer.

Now that I have the time and the continuing curiosity to delve into Apple][DOS, I have the unique opportunity to create my own version of DOS that contains the power and the flexibility I always thought DOS ought to and could have. I call this version of Apple][DOS, DOS 4.3 File Management System, and it requires an Apple][that contains Language Card memory. This document describes my eighth build of DOS 4.3. What a ride I have been on! Why? To see what I could do with this brilliant machine and its magnificent architecture!

2. Brief Overview of DOS 4.3

I know there are a great many ProDOS users in the Apple][community, but I never became interested in ProDOS. The work I did at Hughes Aircraft in the mid 1980's consisted of using assembly language for programming an operating system executive and interface driver routines on Gould SEL 2780, 6780, and 9780 mainframe computers. These computers hosted a proprietary operating system that allowed our team to simulate a radar processor traveling above the earth's surface in virtually real time. In order to accomplish that goal and simulate real time navigation, the computer's file system was essentially flat: each user had their own directory, and these user directories contained no subdirectories. I was very comfortable with the idea of a flat file system and it was very much like Apple's DOS 3.3. I was simply not comfortable with a slew of subdirectories exemplified by Apple's ProDOS. My thought was always "How does one remember the path to follow in order to find anything?" With the advent of the Macintosh computer and later when I became familiar with the UNIX file system, my subdirectory fears vanished and I cannot imagine a modern computer file system without subdirectories. However, I still remain passionate about Apple][DOS and I leave ProDOS to those who are comfortable with that operating system architecture. Though what I have seen of ProDOS recently, I believe it could definitely use a facelift, seriously. I also believe that ProDOS is better suited on a machine with a 16-bit processor much like that found in the Apple //gs.

I am sure many are curious and want to know what is new and different in DOS 4.3, and what makes this version of the DOS File Management System so special. Looking back over my previous build manuals for DOS 4.1, I realized that I should have included this vital build enhancement information with every build, if only for historical reasons. Like, which build did I solve the Track `0x00` utilization quest? Which build did I start labeling volumes? Which build did I solve the `Disk Full` logic error? Taken all together, I have done an incredible amount of research, writing, and software development to reach DOS 4.3, Build 8. And, to say the least, I have done an incredible amount of testing for every function under normal and abnormal (i.e. error) conditions. However small the list of items unique to DOS 4.3 may appear, I have spent countless hours developing and testing those items alone and in concert with the entire DOS 4.3 command repertoire.

DOS 4.3 is specifically designed to reside in Main Language Card memory. Language Card memory begins at address `0xD000`, and it includes two banks of memory from `0xD000` to `0xDFFF` and one bank of memory from `0xE000` to `0xFFFF`. The user has complete freedom to use all memory below `0xBE00` where `HIMEM` is set. The foundation for DOS 4.3 was DOS 4.1H which introduced the `HELP` command. However, I have completely redesigned `HELP` in how it looks and how it works in DOS 4.3, though it provides the same information. I have added a companion command to the DOS `TS` command called `WTS`. `WTS` (Write Track/Sector) allows the user to modify a single byte at a time on any sector of a disk volume. And, DOS 4.3 introduces the DOS `TOUCH` command that will update any file's date and time stamp. Of course, if a file is locked nothing about the file can be changed nor can the file be deleted, unless there is an override available. DOS 4.3 provides that override for the DOS `RENAME`, `TOUCH`, and `DELETE` commands. Thus, DOS 4.3 introduces two additional File Manager opcodes to support `WTS` and `TOUCH`. DOS 4.3 introduces the DOS `PHASE` command which allows the user to set the number of Disk][stepper motor half-phases between adjacent tracks. Finally, DOS 4.3 introduces the `CONFIG` command which gives the user ultimate control over many DOS 4.3 display and input functions. Not that I allowed a single error to reside in any DOS 4.1 build, there are sometimes better programming methodologies. DOS 4.3 does contain many of the DOS 4.1 routines rewritten having a far better design in subverting virtually all possible DOS command programming consequences due to user naiveté. Let's begin with some software design strategies.

3. DOS 4.3 Software Development

In order to design reliable software for a particular machine or platform, one must understand the machine's complete architecture. I believe this design approach is fully applicable to the Apple][computer: either code or data occupies fixed addressable memory where some defined memory locations are reserved for the stack, text, graphics, control, and peripheral slot cards. Code is further restricted in the Apple][by the rather limited 6502-microprocessor Instruction Set. My obvious goal strategy is to design software in such a way as to create the most functionality with the least amount of code and data space. I believe this methodology will yield the highest degree of code effectiveness.

I use Gerard Putter's application Virtual][, Version 9.3, to create my software applications, and that is the platform I use to perform the initial, though simulated testing. Once I am satisfied with a program or a utility operating within the Virtual][application, I transfer the volume image containing that program or utility to an Enhanced Apple //e. I have found some discrepancies between Virtual][and my Enhanced Apple //e particularly in enabling Language Card memory: two successive writes to memory address 0xC083 does **not** write enable Language Card Bank 2 memory in my Enhanced Apple //e as it does in Virtual][. Two successive reads of memory address 0xC083 functions the same in both my Enhanced Apple //e and in Virtual][as they should. I have brought this to the attention of Mr. Putter. Also, Main memory is not initialized at power-up in quite the same way in my Enhanced Apple //e as it is in Virtual][. I believe DOS 3.3 always assumed that an Apple][will power-up with all of page-zero memory set to 0xFF. Virtual][also makes this same assumption. I know I have been caught unaware that all of Auxiliary page-zero memory is not always set to 0xFF at power-up. Therefore, I have included a call to SETNORM during Boot Stage 2 to ensure that page-zero memory location 0x32 is, indeed, set to 0xFF. I have used AUXMOVE to manually "hide" some ProDOS code in Auxiliary memory within Virtual][. The code disappears (is overwritten) when I boot with DOS 4.3. This does not happen in the Enhanced Apple //e: the code can still be safely found in Auxiliary memory after a reboot. Always, always, always make final tests on **real** hardware.

Before beginning any discussion of a complicated subject like a disk operating system or file management system for the Apple][, it is usually easier to understand such a system if each component of that system is shown as part of a Big Picture. That Big Picture is shown in Table I.3.1. Though certainly not to any scale, Table I.3.1 shows how memory is utilized in the Apple][and where the basic hardware and software components are found in Main memory. I exclude any discussion of Auxiliary memory as found in the Apple //e at this time. The basic components shown in Table I.3.1 are the 6502 microprocessor memory requirements, the DOS vectors and routines, text and LORES graphic pages, HIRES graphic pages, DOS file buffers, DOS software manager locations, Soft Switches, peripheral-card memory, Read/Write Track/Sector (RWTS) and HELP routines, Applesoft interpreter, and the ROM and RAM Monitor. The following pages will discuss the Apple][memory utilization in great detail. It may be helpful to refer to Table I.3.1 occasionally in order to fully understand how those details relate to the entire hardware and software management of the Apple][computer by the DOS 4.3 File Management System.

If any of the components shown in Table I.3.1 are unfamiliar, it would be to your advantage now to locate one or more Apple publications and refresh your understanding of that component. Even the *Apple][Reference Manual* that came with my Apple][+ computer contains invaluable information applicable to the entire family of Apple][computers. I even own a few ***SAMS*** Publications that have provided me with enhanced understanding of many of the components shown in Table I.3.1.

Memory Page	Description	Description
0x00	Page-zero variables, pointers, routines, and special addressing modes	
0x01	Stack for the 6502 microprocessor	
0x02	Input buffer, Applesoft interpretation buffer	
0x03	User buffer, DOS vectors and routines	
0x04-0x07	Text or LORES graphics Page 1	
0x08-0x0B	Applesoft program start, Text or LORES graphics Page 2, or available for software	
0x0C-0x1F	Available for software	
0x20-0x3F	HIRES graphics Page 1, or available for software	
0x40-0x5F	HIRES graphics Page 2, or available for software	
0x60-0xBD	Available for software	
0xBE-0xBF	DOS 4.3 HIMEM, DOS 4.3 Language Card interface, DOS 4.3 bootstrap routines	
0xC0	System Soft Switches	
0xC1-0xC7	Peripheral-card ROM memory for slots 1-7, or CX ROM	
0xC8-0xCF	Peripheral-card expansion ROM memory for slots 1-7, or CX ROM	
0xD0-0xDF	Bank 2, ROM Applesoft Interpreter, DOS 4.3 Command and File Managers	Bank 1, DOS 4.3 RWTS and HELP
0xE0-0xEB	ROM Applesoft Interpreter, DOS 4.3 Command and File Managers	
0xEC-0xEF	ROM Applesoft Interpreter, DOS 4.3 working variables and file buffers	
0xF0-0xF7	ROM Applesoft Interpreter, DOS 4.3 file buffers	
0xF8-0xFF	ROM Monitor and RAM Monitor	

Table I.3.1. Apple][Memory Utilization with DOS 4.3

The Apple][computer is truly a brilliant machine and it has a magnificent architecture. I hope you find my presentation of DOS 4.3 vis-à-vis the Apple][computer interesting, enlightening, and useful in view of your own hardware and software experiences with this delightful computer.

4. Page-Zero Utilization

The Instruction Set for the 6502-microprocessor (and the 65C02 processor as well) includes special processor instructions that utilize variables located in the first 256 bytes, or page, of addressable memory, that is, locations `0x0000` to `0x00FF`. I designate this area of memory "page-zero." When Steve Wozniak designed the Apple Monitor, he allocated a number of page-zero locations for its variables and pointers. Similarly, Applesoft, DOS, and virtually all other user assembly language programs use page-zero locations in order to utilize those special instructions. The 6502-microprocessor contains an accumulator, the A-register, and two index registers, the X-register and the Y-register. Page-zero instructions using these registers include load and store instructions, indexed load and store instructions, indexed indirect addressing instructions using the X-register, and indirect indexed addressing instructions using the Y-register. Page-zero wraparound occurs with page-zero indexed addressing and indexed indirect addressing instructions using the X-register and page-zero indexed addressing instructions of the X-register using the Y-register, but not with indirect indexed addressing instructions using the Y-register. Yes, it is a little confusing, but not too complicated.

When developing a user assembly language program, it is critical to select page-zero locations that do not conflict with the Apple Monitor, Applesoft, or DOS depending on whether those ROM and language card applications are important to the user program. Knowing which page-zero locations are used by or critical to resident applications can greatly simplify the selection of unused or available page-zero locations. Because DOS 3.3 supports Integer BASIC, a few page-zero locations were used to process that file type. DOS 4.3 also uses those same page-zero locations for processing the Applesoft `CHAIN` command, for example, and other DOS command enhancements. There are definitely obvious page-zero locations that cannot be used except for how they were intended, like the horizontal and vertical cursor locations `CH` and `CV`, respectively. Then, there are less obvious, rather dubious page-zero locations that are used by some Applesoft commands from `0x00` to `0x1F`. These page-zero locations are fair game for user programs that do not use the Applesoft interpreter or Steve Wozniak's *SWEET16* interpreter. Figure I.4.1 shows all the used and the unused page-zero locations and the applications that use those particular locations according to my references and the best of my ability to decipher the code that uses those locations. The shaded locations in Figure I.4.1 are unused page-zero locations that probably are not used by the Apple //e Monitor or Applesoft, so they are more than likely the better locations to select. Tables I.4.1 and I.4.2 lists all the page-zero locations utilized by DOS 4.3. Table I.4.3 summarizes all the available page-zero locations not utilized by the ROM routines and DOS 4.3. Keep in mind that indirect indexed addressing instructions using the Y-register do require a page-zero byte-pair, so it is even more critical that neither address byte is clobbered by software external to a user's assembly language program.

There are certainly common page-zero locations that all software routines can use as temporary variables and pointers. The 6502-microprocessor is not time-shared and there is no context switching between routines, so if a routine uses some common page-zero locations, it should complete all processing using those locations and not expect to find its results sometime later. Examples of common page-zero locations would be `A1L/A1H` at `0x3C/0x3D`, `A2L/A2H` at `0x3E/0x3F`, `A3L/A3H` at `0x40/0x41`, `A4L/A4H` at `0x42/0x43`, `OPRND` at `0x44`, and the first three bytes of `DSCTMP` at `0x9D:0x9F`. Using these page-zero locations to move or copy data would be safe and not interfere with the Monitor, Applesoft, or DOS processing. Actually, several Monitor routines require that some of these locations just mentioned contain your data before using those routines. The Monitor routine `MOVE` at `0xFE2C` is one such example. It is really up to the user to confirm and verify that the selected page-zero memory locations do not interfere with other routines external to and required by the user software.

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	12 34	134	34	34	34	4					4	4	4	4	4	4
10	4	4	4	4	4	4	4	4	4	4	4	4	4	24	2	3
20	134	134	134	134	13 46	134	14 56	14 56	13 46	13 46	13 456	13 456	14 56	14 56	123 456	123 456
30	14	12	134	12 46	123	12 36	136	136	136	136	123	123	12 345	12 345	123 456	123 456
40	156	156	12 36	12 36	12 36	1	1	1	1	1	56	56	6	6	13	13
50	346	346	4	4	4	4	24	24	24	24	46	6	6	6	4	4
60	4	4	4	4	4	4	4	46	46	46	46	46	46	346	346	346
70	346	4	4	346	346	4	46	4	4	4	4	4	4	4	4	4
80	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
90	4	4	4	4	34	34	4	4	4	4	4	34	34	4	4	4
A0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	46
B0	46	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
C0	4	4	4	4	4	4	4	4	4	4	4	4	4	4		
D0	4	4	4	4	4	4	46		46	6	4	4	4	4	4	4
E0	4	4	4		4	4	4	4	34	34	4					
F0	4	4	4	14	14	4	4	4	4	4						34

Figure I.4.1. Page-Zero Memory Utilization

Key

1 – used by the Monitor
2 – used by the Mini Assembler
3 – used by the Apple //e `CX` ROM
4 – used by Applesoft
5 – used by `RWTS`
6 – used by DOS 4.3

Address	Parameter	Description
0x24	CH	horizontal cursor location
0x25	CV	vertical cursor location
0x26	BUFRADRZ	ROM boot data field buffer address
0x26	TEMPZ	RWTS temporary 8-bit variable
0x27	TEMP2Z	RWTS temporary 8-bit variable
0x28	BASEZ	text screen line address
0x2A	ASPTRSAV	DOS CHAIN array descriptor addresses
0x2A	CURTRKZ	RWTS requested track
0x2B	DRVFLAG	RWTS data-changing drive flag
0x2B	SLOT16Z	boot slot * 16
0x2B	SYNCNT	RWTS format sync byte count
0x2C	DATAFNDZ	RWTS address field address
0x2D	SECFNDZ	RWTS address field sector found
0x2E	TRKFNDZ	RWTS address field track found
0x2F	VOLFNDZ	RWTS address field volume found
0x32	INVFLG	text screen inverse/normal flag
0x33	PROMPT	text screen prompt character
0x35	PAGECNT	boot/initialization DOS image page count
0x36	CSWL	output device handler address
0x38	KSWL	input device handler address
0x3C	ROMTEMPZ	ROM boot temporary 8-bit variable
0x3C	MOTORTIM	RWTS motor on-time 16-bit count
0x3C	A1	general purpose temporary 16-bit variable
0x3D	ROMSECTR	ROM boot requested sector
0x3E	BUFADR2Z	RWTS data field buffer address
0x3E	ODDBITSZ	RWTS temporary 8-bit variable
0x3E	A2	general purpose temporary 16-bit variable
0x3F	SECTORZ	RWTS address field sector
0x40	ROMDATA	ROM boot address field track found
0x40	FILEBUFZ	file context block parameter buffer address
0x40	TRACKZ	RWTS address field track
0x41	ROMTRACK	ROM boot requested track
0x41	VOLUMEZ	RWTS address field volume
0x42	A4	general purpose temporary 16-bit variable
0x42	BUFADRZ	general purpose sector data buffer address
0x44	DIRINDX	VTOC and TSL data index
0x4A	IOBADR	RWTS IOCB buffer address
0x4C	DOSPTR	DOS general purpose pointer address

Table I.4.1. DOS 4.3 Page-Zero Utilization – Part 1

0x50	LINNUM	Applesoft line number 16-bit variable
0x5A	DOSTEMP1	DOS general purpose 8-bit variable
0x5B	DOSTEMP2	DOS general purpose 8-bit variable
0x5C	DOSBUFR	DOS general purpose buffer address
0x67	ASPGMST	Applesoft program start address
0x69	ASVARS	Applesoft simple variables pointer
0x6B	ASARYS	Applesoft array pointer
0x6D	ARYEND	Applesoft end of array pointer
0x6F	ASSTRS	Applesoft end of string storage pointer
0x73	ASHIMEM	Applesoft HIMEM address
0x76	ASRUN	Applesoft RUN flag
0x9D	DSCTMP	Applesoft temporary string descriptor
0xAF	ASPEND	Applesoft end of program address
0xD6	PROTECT	Applesoft program write-protect flag
0xD8	ASONERR	Applesoft ONERR error flag
0xD9	RKEYWORD	DOS R keyword 8-bit variable

Table I.4.2. DOS 4.3 Page-Zero Utilization – Part 2

Start	**End**	**Description**
0x06	0x09	4 bytes free
0x1E	0x1E	1 byte free
0xCE	0xCF	2 bytes free
0xD7	0xD7	1 byte free
0xE3	0xE3	1 byte free
0xEB	0xEF	5 bytes free
0xFA	0xFE	5 bytes free

Table I.4.3. Available Page-Zero Locations Summary

```
           DOS 4.3H
           (c) 2020
 Walland Philip Urbancic Jr


M=4308H P=04 T=DOS 4.3H Demo Disk
B=4308H boot L=0x2020 01/01/20 08:28:48
S=6 D=01 V=000 F=0510 01/01/20 08:28:48
 001 0x12,0x0F HELLO



]
```

This publication describes the process and products I created when I decided to design and program an enhanced Disk Operating System (DOS) for my Apple //e. Wherever I am able, I have included code samples, schematic diagrams, equations, figures, tables, and representative screen shots to help explain what I have created and the reasons why I did so. As in my previous design of an Apple][DOS, i.e. DOS 4.1L and DOS 4.1H that I released in 2018, this has been an incredible journey for me. With DOS 4.3, I have again re-imagined that time when I mostly lived, breathed, and worked on Apple][hardware and software development continuously for a good period of my life many, many years ago. The list that follows are some of the features that I have engineered into the DOS 4.3 File Management System.

DOS 4.3 boots directly into Language Card memory and sets `HIMEM` to `0xBE00`. All five file buffers are fully contained in Language Card memory.

DOS 4.3 supports the full integration of the *Lisa* and *Big Mac* assemblers in Auxiliary Memory. Both assemblers now support the full 65C02 instruction set. *Sourceror* and *Big Mac* both support all *SWEET16* opcodes. My ROM image supports the 65C02, *SWEET16*, and `GARBAGE` based on Bongers' algorithm.

DOS 4.3 supports most Clock Cards which play an integral role in Volume and File timestamps. Volumes may be titled, numbered, locked, and initialized to boot or to store data exclusively. Even track `0x00` is used for data. Files are loaded and saved using an acceleration algorithm.

DOS 4.3 supports Applesoft `CHAIN`, File `DIFF`, File `GREP`, File `LIST`, File Undelete, Sector Display, Sector Modify, Syntactical `HELP`, DOS Processing Control, Variable Management, Half-Phase Tracking, and full lowercase command and argument input.

DOS 4.3 copies files using a modified *FID* that accepts Volume and Phase Number parameters.

DOS 4.3 can read DOS 3.3 Volumes and files.

DOS 4.3 can initialize Volumes with up to 48 tracks having either 16 or 32 sectors per track.

All the source code and all the schematics can be requested through www.applecored.net.

$39.95
ISBN 978-0-578-71547-6
53995>
9 780578 715476