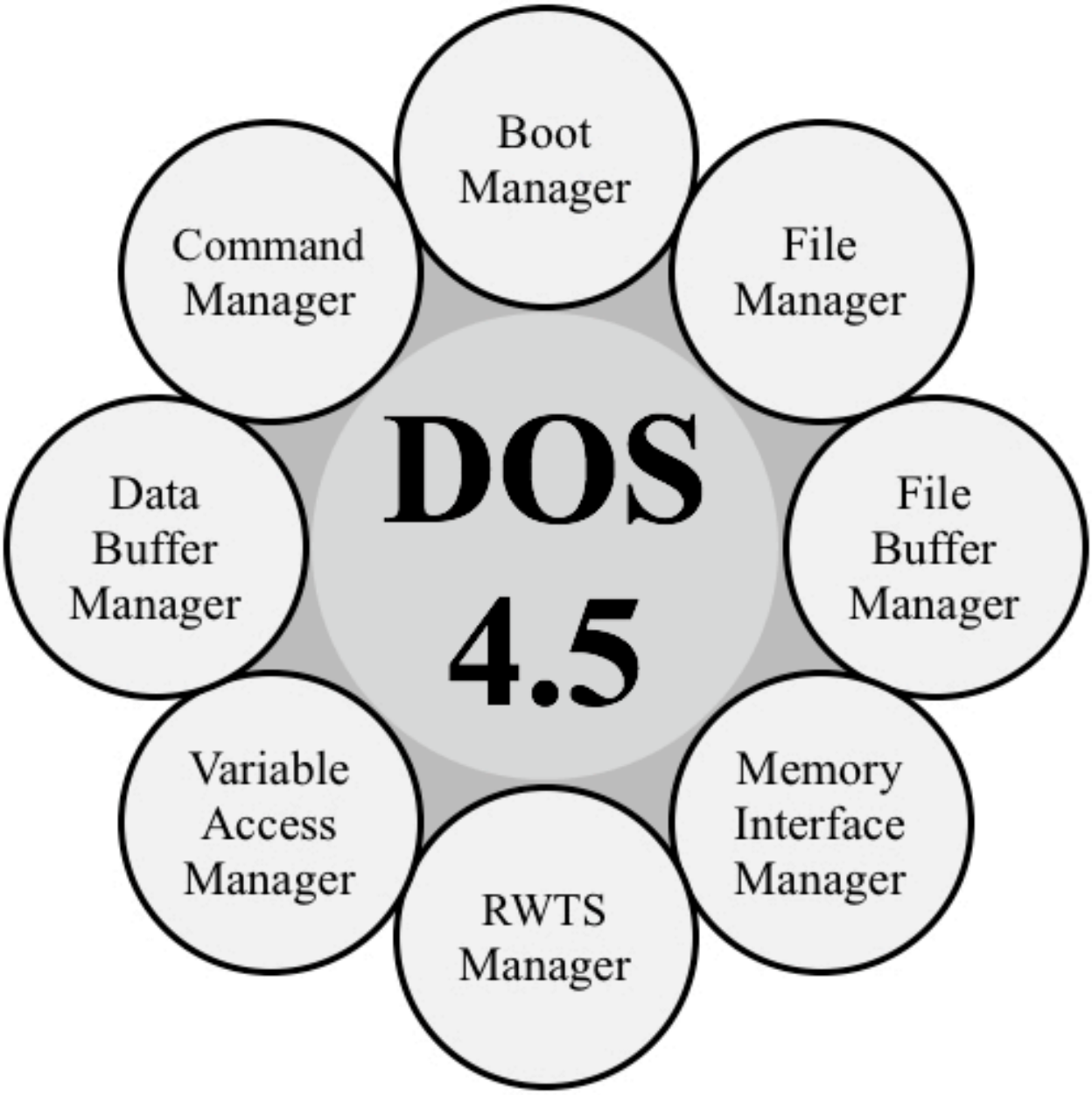
DOS 4.5 File and Volume Disk Management System



Walland Philip Vrbancic, Jr.

DOS 4.5

File and Volume Disk Management System

Walland Philip Vrbancic, Jr.

lulu

DOS 4.5 File and Volume Disk Management System
and this Publication are the
Confidential and Proprietary Intellectual Properties
of
Walland Philip Vrbancic, Jr.

ISBN 978-0-578-38584-6

I am proud to dedicate this Book on the
DOS 4.5 File and Volume Disk Management System
and all of my previous achievements
to my loving Parents Walland and Melba
who continuously nourished my intellectual curiosities
with games, toys, books, education, and unconditional love.

I am truly fortunate for all of the love and support
I have always received from my sister Marile.

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

Excellence is never an accident!
It is always the result of high intention, sincere effort, and intelligent execution;
it represents the wise choice of many alternatives;
choice, not chance, determines one's destiny.

~~~ Aristotle ~~~

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 4.5 File and Volume Disk Management System software 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 4.5 File and Volume Disk Management System software 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 they may result in the loss of recorded data on any storage device connected to your computer. You assume all risks associated with the operation of your computer and the potential loss of your recorded data when using the DOS 4.5 File and Volume Disk Management System software or this Book. If these terms are not acceptable to you, please do not use the DOS 4.5 File and Volume Disk Management System software or this Book.

Walland Philip Vrbancic, Jr., the administrator of www.applecored.net, makes no warranties either expressed or implied with respect to the DOS 4.5 File and Volume Disk Management System software 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 4.5 File and Volume Disk Management System software or the use of information contained in this Book shall be assumed by you, the User. In no event shall Walland Philip Vrbancic, Jr., or www.applecored.net be liable for any direct, indirect, incidental, or consequential damages resulting from any defect, deficiency, or neglect in the DOS 4.5 File and Volume Disk Management System software 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 the loss of recorded data resulting from the use of the information, circuit diagrams, and example software programs contained herein. The author reserves the right to implement any changes and/or any improvements to the DOS 4.5 File and Volume Disk Management System software or to the contents of this publication at any time and without any prior notice to you, the User, or to the general Apple][community.

Preface

When Brian Wiser and Bill Martens discovered my original DOS 4.1 software and documentation at www.applecored.net in 2017, they immediately contacted me and wanted Apple Pugetsound Program Library Exchange (*Call-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 featuring 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 early computer journals would have been crazy exciting at that time, and it would have certainly been a cherished memory for a lifetime. I actually was very close to completing all of the capabilities I wanted in DOS 4.1 when I agreed to have *Call-A.P.P.L.E.* publish the DOS 4.1 Manual for Build 45. I also provided *Call-A.P.P.L.E.* with demo diskette images for both DOS 4.1L and DOS 4.1H. A month or so after Build 45 was published and available for purchase, I completed DOS 4.1 with Build 46. Only the DOS 4.1 Build 46 software and documentation in PDF form are available at www.applecored.net.

I remember telling Wiser and Martens that 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, whereas I had ample memory for DOS 4.1H. Actually, 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 of our negotiations, I warned Wiser and Martens that I could not stop creating more functionality in DOS 4.1, but they were rather insistent on publishing the DOS 4.1 Manual for the Apple][community in its current stage of development. In 2021 I had the opportunity to republish the DOS 4.1 Manual as a book in the format that I originally intended. This publication is Build 46.

My next area of exploration for an 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 new DOS I created to reside in Auxiliary memory. Over the course of several months of significant 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 unique DOS. But this effort was certainly not wasted! I documented what I had learned about Main and Auxiliary memory management and I moved forward to other areas of exploration.

I had already decided to 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, I did not want to develop a companion "Low" memory DOS version in parallel, so I did **not** develop a DOS 4.3L. I simply refer to this new DOS as DOS 4.3. The question then became, can *Lisa* be ported to and function in Auxiliary memory? The answer to that question turns out to be a resounding "*Yes!*" With DOS 4.3 in the memory of the Language Card partition in Main memory and *Lisa* in the memory of the Language Card partition in Auxiliary memory, the user has access to virtually all

of Main memory below `0xBE00` for source code, object code, and the complete symbol list. I saw this exciting configuration as the path to many new and potential possibilities.

Now, if I can relocate *Lisa* to Auxiliary memory, what about doing the same for *Big Mac*? I have to say that that challenge was a bit uneventful because relocating *Big Mac* to the memory of the Language Card partition in Auxiliary 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 capabilities, 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 extraordinarily massive undertaking since I wanted *Sourceror* and *Big Mac* to disassemble/assemble my personal version of the *SWEET16* opcodes. I realized that *Big Mac* could not even assemble its own unique *SWEET16* `EVAL` opcode because the `EVAL` opcode did not even exist. This tells me that Mr. Bredon probably could not even use *Big Mac* to assemble his own *Big Mac* source code unless he possibly hardcoded these undefined opcodes as `byt` statements. I have to confess that there still remain two *SWEET16* branches in my disassembled (and verified) *Big Mac* source code that are absolutely 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 code sequences may be part of `MACRO` handling, something I have had no reason to use. After all is said and done, *Big Mac* and DOS 4.3 complemented each other beautifully.

While moving past DOS 4.3 to begin the development of the DOS 4.5 File and Volume Disk Management System, I discovered even more layers of File Manager functionality that I could code so much more effectively. DOS 4.3 certainly contains all the necessary solutions to properly close files and flush volumes when the DOS `CLOSE` command is issued from within an Applesoft program. Yet, the code resolutions found in DOS 4.3 still do not exhaust my list of all the additional capabilities I wanted in my final release of an Apple][DOS. All of the capabilities contained in DOS 4.5 are detailed in Section I.2.

I know the user will discover many, if not all of the fascinating developments I have included in the DOS 4.5 File and Volume Disk Management System. The user will be left wondering how he or she was able to accomplished anything useful 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 4.5 Assembly Language Routines 171

V. DOS 4.5 Operational Environment 193

This page intentionally left blank.

List of Figures

List of Tables

DOS 4.5 File and Volume Disk Management System

I. Designing Another New DOS

This publication describes the process and the products I created when I decided to design and program another 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 many of the reasons why I did so. As in my previous designs of an Apple][DOS, i.e. DOS 4.1 and DOS 4.3, this has been an incredible journey for me. With DOS 4.5 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][+ computer, my initial system included an Apple][Language Card, an Apple Disk][Drive with a Disk][Interface Card, an Amdek color monitor, and an Epson MX100 printer with a Grappler+ Printer Interface card. During those early years I designed and built my own Apple][peripheral slot cards, I made electrical and hardware modifications to my Apple][+ motherboard and keyboard, and I wrote a substantial number of software programs initially using Applesoft BASIC (Applesoft hereafter) and then a few months later, I wrote software in 6502 assembly language. I soon acquired a Videx UltraTerm video display card and a Microsoft Z80 Softcard. With the Z80 Softcard I was able to write complex 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 cards to my system, a Johnathon Freeman Designs (JFD) Parallel Printer Buffer, and an Axlon RAM Disk 320 with its interface card.

I used C language in my professional programming career for the design and 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 version of DOS 3.3 for the Apple][series of computers. I thought the last DOS 3.3 version was published on August 25, 1980. Then I recently came across another DOS 3.3 version published years later on January 1, 1983. That later DOS contains even more patches for the DOS `APPEND` command and 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 its structure and in its implementation. Apparently, very little attention was given to software design and review. It appeared to me that Apple made a strong push to release *something or anything* to consumers and vendors in

order to begin marketing software products on diskettes and hardware products to read and write those diskettes. And history does reveal that Apple Computer did outsource DOS and contracted for it to be delivered within thirty-five 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/writer, 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, the DOS 4.5 File and Volume Disk Management System, and it requires an Apple][that contains memory in the Language Card partition for its complete operation. This publication describes my fifth build of DOS 4.5. What a ride I have been on! Why? To see what I could do for this brilliant machine and its equally magnificent architecture! I hope that you find my journey into DOS 4.5 as fascinating as I found developing it.

2. Overview of the Improvements and Enhancements in DOS 4.5

I know there are a great many ProDOS users in the global Apple][community, but I never became at all 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 hardware 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 the hardware of a Radar Digital Processor (RDP) 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: every software developer had their own directory, and these user directories contained no subdirectories. I was very comfortable with the idea of a flat file system as 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 recall 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 far better suited on a machine with a 16-bit microprocessor 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 4.5, and what makes this version of the DOS File and Volume Disk Management System so special. Looking back over my previous publications of DOS 4.1 and DOS 4.3, I realized that I should have included this enhancement information with every version and for each software build, if only for historical reasons. Like, which version and build did I solve the Track `0x00` utilization quest? Which version and build did I start labeling volumes? Which version and 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 the level of perfection that is contained in DOS 4.5 with Build 5. And, to say the least, I have done an incredible amount of unit testing for each and every logical function under normal and abnormal (i.e. error) conditions. However small the list of improvements and enhancements unique to DOS 4.5 may seem, I have spent countless hours developing and testing those improvements and enhancements alone and in concert with the overall DOS 4.5 command repertoire and its system functional capabilities.

Module	Description of Improvement or Enhancement
CMD1	Move CMDINDX to keyboard variables and clear keyboard variables before, not after parsing a new DOS command. No need to range check SV and USER parameters.
CMD2	Modified TS to read sector data into caller's specified buffer. Added ENTRMON routine that calls INITPTRS before entering RAM Monitor. Removed COPYVALS after moving CD to File Manager with CDHNDLR. Moved DOADRINC to CMD2.
CMD3	Rewrote GREP so that it no longer requires a character string marker for a multiple-word character string search, and less code was required (Aha! Moment)
CMD4	FMDRVR now copies VALSPHAS to FMPHASE, CONFIG is set to default if R keyword is included, PHASE is set to default if R keyword is included, MAXFILES is set to default if R keyword is included, SAVALRD writes CONFIG, PHASE, and MAXFILES values into the Value Read Buffer if DOS is in the RUN mode, SV writes its value into the Value Read Buffer if DOS is in the RUN mode (Aha! Moment), CD uses File Manager to read VTOC for CONFIG value, PRTSDV now prints volume lock status.
MNGR1	External File Manager entry uses FILALC/NOFILALC values to set KEYWORD1 for file allocation control, now FMTBLJMP uses memory placement of File Manager routines to selectively copy FM Context Block SDV values to FMWORK since Byte Offset and Range Length are overloaded values in FMWORK.
MNGR2	TOUCH only copies VALSCNFG to DOSCONFG if R keyword is included to save the DOS configuration, DELETE handler rewritten and its logic reordered, CATALOG handler does not show build information if DOSVRSN < 0x33 or DOSBUILD = 0, FREESECT now clears SECBTMAP before calling RORBITMP, not after. Added CDHNDLR.
MNGR3	ALLOCSEC modified to perform up to 4 scans of the VTOC bitmap, ALLOCNTR moved to CMDVALS.
DATA1	Moved CMDUSER before CMDHELP, set KWRANGEH for drive to 81.
BUFR1	Moved CMDINDX after MONVAL in keyword variables and put ALLOCNTR after CMDLNIDX where CMDINDX was, removed TRKNUMBR from FMWORK, set VALRDBUF to 8 bytes. File buffer is now 0x0245 bytes in size. Added 7-byte SCRCHTBL table (Aha! Moment) and adjusted SPAREBUF size.
BUFR2	Removed WATRKNUM in WORKAREA (parallel variable to TRKNUMBR).
RWTS1	If slot number index to SCRCHTBL is zero, RWTS scratchpad RAM locations for track and phase are cleared for drives 1 and 2, and calls BLDNIBL when RWTS is called the first time for that slot number (Aha! Moment). BLDNIBL dynamically builds RDNIBL and WRNIBL tables in same page with NBUF2.
RWTS2	PHASE value better managed and DISKFMT now initially formats track 0x00, reduces SYNCNT, then formats entire volume starting with track 0x00.
MNGEXVAL	Modified this function to return the Y-reg incremented twice when reading/writing 16-bit values and once when read/writing 8-bit values found in the CMDVALS and FMWORK Data structures. Removed access to the INITVALS Data structure.
MNGEXUSR	Removed the call to CLRVALS from this function. The call to CLRVALS is now properly made by DOSINIT just prior to the COLDSTRT entry point.
HELP	Any key press other than ESC, RTN, or arrow keys select the display of the DOS 4.5 Management screen.
USER	The USER handler now resides in Main memory, its call to the USER address returns to Main memory, and finally USER re-enables RAM memory in the Language Card partition to complete its processing.

Table I.2.1. Major Improvements and Enhancements to DOS 4.5

DOS 4.5 is designed to reside either in Main memory or in the memory of the Language Card partition. The version of DOS 4.5 that resides fully in Main memory is DOS 4.5L, and its executable code begins at 0x9F00, its variables and data buffers begin at 0x9A8A, and its two file buffers begin at 0x9600 where HIMEM is set, the very same memory location as in DOS 3.3. In comparison, DOS 4.1 sets HIMEM to 0x9625 when MAXFILES is set to three file buffers. Thus, the additional code that comprises DOS 4.5L is essentially the size of one file buffer. Unlike DOS 3.3 or DOS 4.1L, DOS 4.5L is a single, continuous object code image that is 0x2100 bytes in size. On the other hand, the version of DOS 4.5 that resides in the memory of the Language Card partition is DOS 4.5H, and it occupies both

banks of memory from `0xD000` to `0xDFFF` and the remaining memory of the partition from `0xE000` to `0xF7FF`. The Apple][user has complete freedom to use all of Main memory below `0xBE00` where `HIMEM` is set.

The foundation for DOS 4.5 is DOS 4.3H which includes the `HELP` command. In order to better appreciate all of the improvements and the enhancements I have added to DOS 4.5, I have tabulated those changes according to each software module as shown in Table I.2.1. Please realize that nothing in DOS 4.3 was removed in order to provide the code space required by all of these improvements and enhancements in DOS 4.5. I simply spent a sizable amount of time and effort re-working certain modules to yield the same or better functionality in far less code space. I can truthfully say that I experienced several Aha! Moments that certainly assisted in making these improvements and enhancements to DOS 4.5 possible.

3. DOS 4.5 Software Development Strategies

Let's begin with some software design and development strategies. In order to design reliable and powerful software for a particular machine or platform, one must understand the complete architecture of that machine. I believe this design approach is fully applicable even 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 using the least amount of code and data space. I believe this methodology yields the greatest range of code effectiveness.

I highly recommend obtaining and referring to a number of reference publications for the Apple][hardware. I have used the Apple][*Reference Manual*, the Apple //e *Reference Manual*, the Apple //e *Technical Reference Manual*, the Apple][*The DOS Manual* Disk Operating System, *Beneath Apple DOS* by Don Worth and Pieter Lechner, *Understanding the Apple][* by Jim Sather, and *What's Where in the APPLE A Complete Guide to the Apple Computer* by William F. Luebbert to obtain much of my understanding in how the Apple][hardware functions. I have also referred to the *APPLESOFT][* manual to obtain my understanding of Apple's Applesoft programming language. It is an absolutely required reference manual to have in order to learn that BASIC programming language. These references provides the reader a fairly complete understanding of the architecture of the Apple][computer as well as how to create software programs using the Applesoft language. No reference manual is perfect and people do make mistakes, however these manuals contain very few errors.

In order to study how other people have approached the hardware architecture of the Apple][in creating their software applications, it is necessary to obtain their source code or generated source code from their object code, the code that actually executes from within the Apple][memory. Rarely have I ever found published source code. Glen Bredon designed *Sourceror* as a subsidiary tool to his assembler *Big Mac* that creates *Big Mac* source code files from assembly language object code files. *Big Mac* can also save source code as text files so that *Big Mac* source code can be migrated to other assemblers like *Lisa*.

For my assembly language programming I use Gerard Putter's application Virtual][, Version 9.3, to create my software applications, and that is the platform I use to perform my initial, though simulated software testing. Once I am satisfied with a software program or utility operating within the Virtual][

environment, I transfer the volume image containing that software program or utility to an Enhanced Apple //e. I have found some discrepancies between Virtual][and my Enhanced Apple //e particularly when I am enabling memory in the Language Card partition: two successive *writes* to memory address `0xC083` does **not** write enable Bank `2` in the Language Card partition in my Enhanced Apple //e as it does in Virtual][. Two successive *reads* of memory address `0xC083` function the same in both my Enhanced Apple //e and in Virtual][to enable Bank `2` memory. 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 assumes that an Apple][powers up with all bytes in page-zero memory set to `0xFF`. Virtual][also makes this same assumption. I know I have been caught unaware that all bytes in Auxiliary page-zero memory are 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 *stash* some ProDOS code in Auxiliary memory from within Virtual][. The code disappears (becomes overwritten) when I then boot into DOS 4.5. This does not happen in the Enhanced Apple //e: the ProDOS code or any stashed code can still be safely found in Auxiliary memory even after a DOS reboot. Always, always, always make final tests using **real** hardware.

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	
`0x60–0xBF`	Available for software	
`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 routines	Bank `1`
`0xE0–0xF7`	ROM Applesoft Interpreter routines	
`0xF8–0xFF`	ROM Monitor routines	

Table I.3.1. Apple][Memory Utilization

Before beginning any discussion of a complicated subject like a file and volume disk 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 the following three tables, Tables I.3.1, I.3.2, and I.3.3. Though certainly not to any particular scale, Table I.3.1 shows how memory is utilized in the Apple][and where the basic Apple][system hardware and software components can be found in

Main memory and in the memory of the Language Card partition which is shown in the more shaded bottom lines in each of these three tables. 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, system Soft Switches, peripheral-card and CX ROM memory, where the ROM Applesoft interpreter is found, and where the ROM Monitor resides. If any of the components shown in Table I.3.1 are unfamiliar, it would be to your advantage now to locate and study one or more of the above referenced publications to refresh and increase 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
0x00–0x03	System, 6502-microprocessor memory utilization	
0x04–0x07	Text Page 1	
0x08–0x95	Available for software	
0x96–0x99	DOS 4.5L HIMEM, DOS 4.5L file buffers	
0x9A–0x9E	DOS 4.5L working variables, workarea buffer, VTOC and catalog buffers, nibble buffers	
0x9F–0xBE	DOS 4.5L Command and File Manager, RWTS, and all other software routines	
0xBF	DOS 4.5L bootstrap routines	
0xC0	System Soft Switches	
0xC1–0xCF	Peripheral-card ROM memory for slots, CX ROM	
0xD0–0xDF	Bank 2 ROM routines	Bank 1
0xE0–0xF7	ROM routines	
0xF8–0xFF	ROM Monitor routines	

Table I.3.2. Apple][Memory Utilization with DOS 4.5L Installed

Memory Page	Description	Description
0x00–0x03	System, 6502-microprocessor memory utilization	
0x04–0x07	Text Page 1	
0x08–0xBD	Available for software	
0xBE–0xBF	DOS 4.5H HIMEM, DOS 4.5H Language Card partition software interface, DOS 4.5H bootstrap routines	
0xC0	System Soft Switches	
0xC1–0xCF	Peripheral-card ROM memory for slots, CX ROM	
0xD0–0xDF	RAM Bank 2, DOS 4.5H Command and File Managers	RAM Bank 1, DOS RWTS
0xE0–0xE6	RAM DOS 4.5H Command and File Managers	
0xE7–0xEC	RAM DOS 4.5H working variables and workarea buffer	
0xED–0xF7	RAM DOS 4.5H file buffers	
0xF8–0xFF	RAM Monitor routines	

Table I.3.3. Apple][Memory Utilization with DOS 4.5H Installed

Table I.3.2 is similar to Table I.3.1 except that the 6502-microprocessor and ROM components have been diminished from view in lieu of showing visually where DOS 4.5L and its components are placed in memory. DOS 4.5L is typically configured to have two file buffers, and those buffers begin at `0x9600` and end at `0x9A89` in memory. From `0x9A8A` to `0x9EFF` is where the working variables and workarea buffer, the `VTOC` and catalog buffers, and the nibble buffers, the write translate, and the read translate tables are defined. The DOS 4.5L software routines reside in the continuous span of memory from `0x9F00` to `0xBFFF`.

Table I.3.3 is also similar to Table I.3.1 and it, like Table I.3.2, shows visually where DOS 4.5H and its components are placed in memory. In order to manage the DOS 4.5H routines located in the memory of both banks in the Language Card partition, a set of software interface routines that control the utilization of memory in the Language Card partition is located in Page `0xBE`. Page `0xBF` contains the DOS bootstrap routines which are similar in nature to those same routines found in DOS 4.5L.

The following sections discuss the utilization of Apple][memory in great detail. It may be helpful to occasionally refer to Tables I.3.1 to I.3.3 in order to fully understand how that memory utilization relates to the entire hardware and software management of the Apple][computer by either version of the DOS 4.5 File and Volume Disk Management System. The Apple][computer is truly a brilliantly designed machine and it has an equally magnificent architecture. I hope you find my presentation of the Apple][computer vis-à-vis DOS 4.5 interesting, enlightening, and useful in view of your own hardware and software experiences with this delightful machine.

4. Page-Zero Utilization

The Instruction Set for the 6502-microprocessor (as well as the 65C02-microprocessor) includes certain microprocessor 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 to be *page-zero*.

When Steve Wozniak designed the Apple ROM Monitor (a collection of low-level software routines), he allocated a number of page-zero locations for its variables and pointers. Similarly, Applesoft, DOS, and virtually all other assembly language programs use page-zero locations in order to utilize those specific instructions. The 6502-microprocessor contains an accumulator called the A-register and two index registers called 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 indexed and indexed indirect addressing instructions using the X-register and indexed addressing instructions of the X-register using the Y-register, but page-zero wraparound does **not** occur with indirect indexed addressing instructions using the Y-register. Yes, initially, addressing modes can certainly be a little bit confusing.

When developing a new assembly language program, it is critical to select page-zero locations that do not conflict with the Apple ROM Monitor, Applesoft, or DOS depending on whether those ROM applications and the applications that reside in the memory of the Language Card partition are important to the new program. Knowing which page-zero locations are used by or critical to ROM and 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 are used to process that file type. DOS 4.5 also

uses those same page-zero locations for processing the Applesoft `CHAIN` command, for example, and many 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 from `0x00` to `0x1F` that are used by some Applesoft commands. These page-zero locations are fair game for new programs that do not use the Applesoft interpreter or Steve Wozniak's *SWEET16* interpreter. Figure I.4.1 shows all of the used and the unused page-zero locations, and the Key defines those applications that use those particular locations according to my references and the best of my ability to decipher the ROM routines that make use of page-zero memory. The shaded areas in Figure I.4.1 are unused page-zero locations that are most likely not used by the Apple //e Monitor or Applesoft, so they are more than likely the better page-zero locations to select. Table I.4.1 summarizes all of the available page-zero locations that are not utilized by the ROM routines and by DOS 4.5 shown in Figure I.4.1. Keep in mind that indirect indexed addressing mode 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 new assembly language program.

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	1234	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	1346	134	1456	1456	1346	1346	13456	13456	1456	1456	123456	123456
30	14	12	134	1246	123	1236	136	136	136	136	123	123	12345	12345	123456	123456
40	156	156	1236	1236	1236	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 ROM 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 4.5

Start	End	Description
0x06	0x09	4 bytes 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.4.1. Available Page-Zero Locations Summary

Tables I.4.2 and I.4.3 list all of the page-zero locations utilized by DOS 4.5 and defined in the DOS 4.5 source code file INCL.L. 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, that routine should complete all calculations and processing using those locations and not expect to find those same 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 0x45, and the first three bytes of DSCTMP at 0x9D:0x9F. As is shown in Tables I.4.2 and I.4.3, all of these page-zero locations are defined in the DOS 4.5 file INCL.L. Using these page-zero locations to move or copy data would be safe and not interfere with the ROM Monitor or Applesoft processing. Actually, several ROM Monitor routines require that some of these page-zero locations just mentioned contain your data before using those routines. The ROM Monitor routine MOVE at 0xFE2C is one such example that uses A1L/A1H and A2L/A2H to move data in memory. 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 any new software developed by a user.

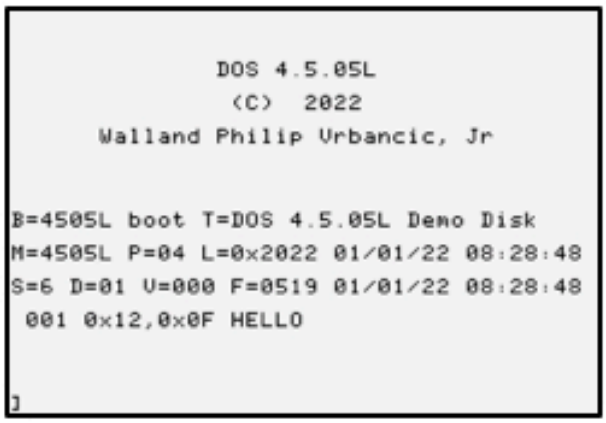
Address	Parameter	Description
0x24	CH	horizontal cursor location
0x25	CV	vertical cursor location
0x26	BUFRADRZ	ROM firmware boot data field buffer address
0x26	TEMPZ	RWTS temporary 8-bit variable
0x27	TEMP2Z	RWTS temporary 8-bit variable
0x28	BASEZ	text screen line address
0x2A	ASPTRSAV	DOS CHAIN array descriptor addresses
0x2A	CURTRKZ	RWTS requested track
0x2B	SLOT16Z	boot slot * 16
0x2C	DRVFLAG	RWTS data-changing drive flag
0x2C	ADRDATMK	RWTS address/data mark
0x2C	ADRFIELD	RWTS sector address field array
0x2D	SECFNDZ	RWTS sector address field sector found
0x2E	TRKFNDZ	RWTS sector address field track found
0x2F	VOLFNDZ	RWTS sector address field volume found

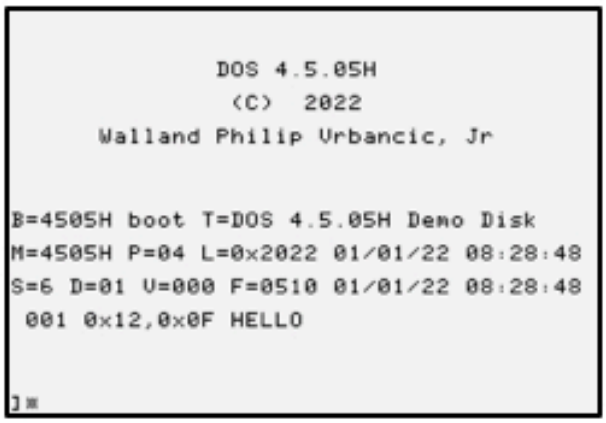
Table I.4.2. Page-Zero Utilization in DOS 4.5 – Part 1

Address	Parameter	Description
0x32	INVFLG	text screen inverse/normal flag
0x33	PROMPT	text screen prompt character
0x34	PHASE	RWTS requested phase number
0x35	PAGECNT	boot/initialization DOS image page count
0x35	SAVXYREG	save X-reg or Y-reg 8-bit variable
0x35	SYNCNT	RWTS synchronization byte count
0x36	CSWL	output device handler address
0x38	KSWL	input device handler address
0x3C	ROMTEMPZ	ROM firmware 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 firmware 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 firmware boot address field track found
0x40	FILEBUFZ	file context block parameter buffer address
0x40	TRACKZ	RWTS address field track
0x41	ROMTRACK	ROM firmware 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
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 16-bit variable/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 character string storage pointer
0x73	ASHIMEM	Applesoft HIMEM address
0x76	ASRUN	Applesoft RUN flag
0x9D	DSCTMP	Applesoft temporary character string descriptor data
0xAF	ASPEND	Applesoft end of program address
0xD6	PROTECT	Applesoft program write-protect 8-bit flag
0xD8	ASONERR	Applesoft ONERR 8-bit error flag
0xD9	RKEYWORD	DOS R keyword 8-bit variable

Table I.4.3. Page-Zero Utilization in DOS 4.5 – Part 2

Index





This publication describes the process and the products I created when I decided to design and program an enhanced Disk Operating System (DOS) for my Apple][+ and for my Apple //e to manage files and volumes. Wherever I am able, I have included source 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 designs of an Apple][DOS, i.e. DOS 4.1L and DOS 4.1H that I released in 2018 (and again in 2020) and DOS 4.3H that I released in 2020, this has been an incredible journey for me. With DOS 4.5, 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 4.5 File and Volume Disk Management System.

DOS 4.5L boots directly into Main memory and sets `HIMEM` to `0x9600` whereas DOS 4.5H boots directly into Language Card memory and sets `HIMEM` to `0xBE00`. Unlike the file buffers in DOS 4.5L, all five file buffers in DOS 4.5H are always available and they are fully contained in Language Card memory.

DOS 4.5 supports the full integration of the *Lisa* and the *Big Mac* assemblers into Auxiliary Memory. Both assemblers now support the full 65C02 instruction set. *Sourceror* and *Big Mac* are fully integrated and they both support all *SWEET16* opcodes. My `C0:FF` ROM image is totally compatible with DOS 4.5 and this image supports the 65C02 processor, *SWEET16*, and `GARBAGE` which is based on Bongers' algorithm. A fix for `HLIN` now draws perfect `HIRES` line graphics for my celebrated *ICON Maker* tool that can be paired with my `HIRES` *CHAR Editor* and *LENGTH* tools for stunning graphic presentations as those used in *BFI*.

DOS 4.5 supports all Clock Cards which plays a vital role in Volume initialization and the management of File timestamps. Volumes may be dated, titled, numbered, locked, and initialized to boot or to store data exclusively. Even track `zero` is used for data. Files are loaded and saved using an acceleration algorithm.

DOS 4.5 supports Applesoft `CHAIN`, File `DIFF`, File `GREP`, File `LIST`, File Undelete, Sector Display, Sector Modify, a programmable and argument-passing `USER` command, Variable Management, Syntactical `HELP`, DOS Processing Controls, Half-Phase Tracking, and full lowercase command and argument input.

DOS 4.5 can read DOS 3.3 Volumes and files.

DOS 4.5 can initialize Volumes with up to 48 tracks having either 16 or 32 sectors per track.

The utilities, tools, schematics, and Excel and Edraw files can be requested through www.applecored.net.

$59.95
ISBN 978-0-578-38584-6
55995>
9 780578 385846