The DOS 4.1 Manual

Disk Operating System for the Apple II

Written & Programmed by:

Walland Philip Vrbancic, Jr.

Produced by:

Brian Wiser & Bill Martens

Apple PugetSound Program Library Exchange

The DOS 4.1 Manual: Disk Operating System for the Apple II

Published by Apple Pugetsound Program Library Exchange (A.P.P.L.E.)

www.callapple.org

Paperback ISBN: 978-0-359-39684-9

ACKNOWLEDGEMENTS

DOS 4.1 was Programmed and Designed by Walland Philip Vrbancic, Jr.

The Cover was designed by Brian Wiser.

PRODUCTION

Brian Wiser → Cover Design, Production
Bill Martens → Production
Walland Philip Vrbancic, Jr. → Documentation, Programming

DISCLAIMER

About Walland Philip Vrbancic, Jr.

I decided to join the team at Rockwell, in the Space Shuttle Simulation Laboratory about five months before the launch of STS-1, changing my hospital scrubs for a coat and tie, and a whole lot more money. About three months after I was hired at Rockwell, another engineer was hired, a Computer Science Engineer, to join the ranks of Initialization Engineers. He and I had the daunting task of initializing the computers and electronics that comprised the simulation of a Space Shuttle from Main Engine Cutoff (MECO) to landing at a few selected American landing sites. Our computers included a PDP-11 and two Xerox mainframes that were programmed using front-panel rocker switches: the Sigma 5 had 16 KB and the Sigma 9 had 64 KB of magnetic core memory. We used Hollerith cards to insert faults into the shuttle's General Purpose Computers (GPCs) and we used a color Eidophor projector to project visual images of our landing site runways into a shuttle cockpit simulator in which the astronauts trained. We used Nova computers by DEC and DEC word processing software to generate all required documentation, and to play *Adventure* I might add.

My colleague was an early Apple II owner when Integer BASIC was initially available in ROM. The following year Rockwell offered a home computer purchase program providing the choice between an IBM PC and the Apple II Plus. My colleague strongly encouraged me to request the Apple computer, and he assisted me in selecting the monitor, disk drive, and printer accessories. The total cost was a lot of money for me but Rockwell loaned me the money and paid the bill, and I repaid the interest-free loan through weekly payroll deductions. Thus began my dream of having my own personal computer.

I became fascinated with all aspects of the Apple II Plus computer, and I wanted to incorporate it into my Master's degree studies. My assigned advisor was analyzing tomographic reconstructions of the human spinal column and he thought perhaps I could assist him. He wanted to be able to make measurements between any two locations within the computer images even after rotating or enlarging the images. I was tasked with developing the Fortran programs that could be launched on a Microsoft Z80 peripheral slot card in an Apple II Plus that would provide him with these capabilities. I found an ingenious way to reduce the size of the three-dimensional rotational matrix in order to accelerate data processing and the mapping of those results to the screen. My professor was very pleased with my progress.

However, I was becoming increasingly interested in high-speed graphics animation and the only way I would learn that technology was to work for Ken Williams at Sierra On-Line. I terminated my work on my Master's degree, gave notice to Rockwell, packed my bags, and moved to Oakhurst, California. At Sierra I assisted a colleague in migrating *ScreenWriter* to the newly marketed Apple IIe, I wrote all the I/O routines and ICON drawing routines for *HomeWord Speller*, and I nearly finished *Goofy's Word Factory*, a children's game to teach English grammar. Williams had a license to display certain Disney characters on a computer screen per approval by Disney for visual likeness and color. I would have finished the product if the designer of the game (Williams' brother) could have developed the third game feature in a timely fashion. He apparently could not do so before I secured a position at Hughes Aircraft Company back in Los Angeles. I stayed all of 18 months at Sierra and I did utilize Williams' high-speed graphics animation algorithms in *Goofy*, which I had to redesign in order to include collision detection on a dithered background. No other computer game could detect collisions on a dithered background at that time. Williams was impressed, and it was really hard to impress Williams.

The major observation I made at Hughes Aircraft was how different the culture was to the culture I experienced at Rockwell. At Rockwell I found it exceedingly difficult to have anyone who had written a software tool explain to me how that tool worked and the algorithms the tool utilized, or exploited.

When I was tasked to migrate a software tool from Fortran to C language at Rockwell, I found some incorrect logic that affected the final data output. Given certain input parameters this tool could calculate a three-dimensional corridor in space and either interpolate points within or extrapolate points outside that corridor. I presented my findings to its author showing how I could insert the same incorrect logic into the C code and generate the same wrong data output. He told me to keep the incorrect logic and not disclose my findings. I refused.

This was totally unthinkable, and this would have never happened at Hughes. In fact, CIP awards were given to engineers for finding such errors in software and bringing those errors to the attention of management. The Hughes culture encouraged the aggressive sharing of knowledge and giving reward for making improvements, not the self-preservation culture of Rockwell where knowledge was thought to be job security and not to be shared, but kept secret. Hughes certainly provided me a great opportunity within the Digital Simulation Laboratory where I learned about real time executive software executing on Gould SEL mainframe computers (9720, 9760, and 9780), MIL-STD-1553 communication protocol, and real time software interface drivers to a host of various external data processors. Our purpose was to create a digital time frame in order to simulate in real time the environment for a tactical Radar Digital Processor flying above the surface of the earth.

In 1990, I returned to Rockwell believing that my knowledge in real time executive software executing on SEL mainframe computers would be my passport to a nice software engineering career closer to where I lived. How I regret this major blunder in judgment because my employment at Rockwell was terminated a few years later. I had co-authored a White Paper outlining the risks associated with using off-the-shelf RISC processors in certain applications, and the response from my colleagues was very unfavorable. Fortunately, my former Hughes management was able to reinstate my position and I was tasked to gain expertise in real time data collection software for tactical radar systems.

Hughes tactical radar systems are programmed to operate in many different modes depending upon various situations and needs faced by the pilot of a military aircraft. During the development of a radar mode, its processing is heavily instrumented which generates a large amount of output data as the mode progresses through its various processing stages. It is critical to capture all that generated data in order for the mode developer to ensure and verify the mode is behaving as expected and is generating data according to pre-established boundaries, much like comparing the data to some gold standard. My task was to capture all the in-phase and Quadrature (I/Q) components of radar data in real time, package the data according to source and timestamp, and save the resulting files to some recording device.

It is important to understand that there are many independent sources of data in a radar system whose timestamps are totally asynchronous. At a later time the data in those files would be analyzed to determine if, in fact, the processing modes operated as expected. Physically collecting this I/Q data during real tactical maneuvers was quite a challenge, and recorders designed to operate in this environment were costly. Preparing for a data collection session involved securing a military aircraft, a flight crew, a ground crew, and people to securely bring the recorded data back to my secure lab. This certainly added to my responsibilities, and my mantra was to neither add, subtract, nor modify any data word or data bit while that data was in my immediate possession and while my software algorithms processed that data.

I was thoroughly vetted and held maximum-security clearances that allowed me to process data from many different and independent programs not only here in Los Angeles, but also in other locations, even out of state. The general data collection software engines I began designing in the unclassified world served as my software library for every classified program I worked. Perhaps I was simply in the right place at the right time that steered my career to become the sole resident expert in Transcription Software, that is, to process, encrypt, and store in real time at least a terabyte of data every second. Or, perhaps I was in the right place at the right time that allowed me to develop a task beyond its envisioned potential. There is a direct ancestral linkage between my unclassified software library of tools, routines, and transcription engines and every single classified program with which I was associated that required my tools, engines, and expertise.

I was practicing "code reuse" light-years before it became a topic that some managers thought could reduce software development costs. Really? Initially I was given the opportunity to host my Transcription Software on a newly acquired SGI Origin 300 having four bricks, or 16 CPUs. "Code reuse" made this task fairly straightforward, thus demonstrating the machine's practicality in their feasibility study. After a fact-finding tour at the SGI facilities at Mountain View, California, I was given the momentous task of designing a Transcription Software engine for an SGI Origin 3000 having eight bricks (32 CPUs) running IRIX, and using Big Endian memory management. This turned out to be one of my greatest achievements. Little did my management know how effortlessly I could build my transcription engines by this time primarily because of "code reuse."

I was extremely fortunate to have had one very intelligent manager who casually asked me to think about the possibility of building a digital playback system. Such a system did not exist. Some had tried building an analog playback system years earlier with very little success. Instead of "analyzing" the collected instrumented data, one could observe how the simulated Radar Digital Processor behaved when the high-speed (I/Q) data and the slow-speed (environment) data is injected back into its system. A few months later I presented my first digital playback recorder and pre-processing system, my last and greatest achievement for Raytheon. I was given the unique privilege to design and build a second digital playback recorder and pre-processing system for another classified program. As for the program that used my first digital playback recorder and pre-processing system, that next program saved countless hours of analysis time and mission costs before I scheduled my overdue retirement.

A few years after I retired I was presented with an astonishing diagnosis that seemed to explain the idiosyncrasies I have displayed my entire life as far back as elementary school: I may have been living with Asperger's. Indeed, how does one know what is normal – that which falls under the umbrella of a Gaussian curve? We are all volitional beings and our behavior is internal to each of us. Our brain is composed of carbon-based synapses whose billions of connections compose the very person and personality we have become or have allowed ourselves to become. It is simply miraculous that any of our species reach total fulfillment of their dreams. I like to think I have come closer than most in reaching many of my aspirations.

I have the time and the continuing curiosity to delve into Apple II DOS now, and I have the 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 my version of Apple II DOS, DOS 4.1. And this is my 46th build of DOS 4.1 with more to come at www.applecored.net. What a ride I have been on! Why? To see what I could do for this wonderful machine and its magnificent architecture!

Table of Contents

List of Figures

List of Tables