Attribution

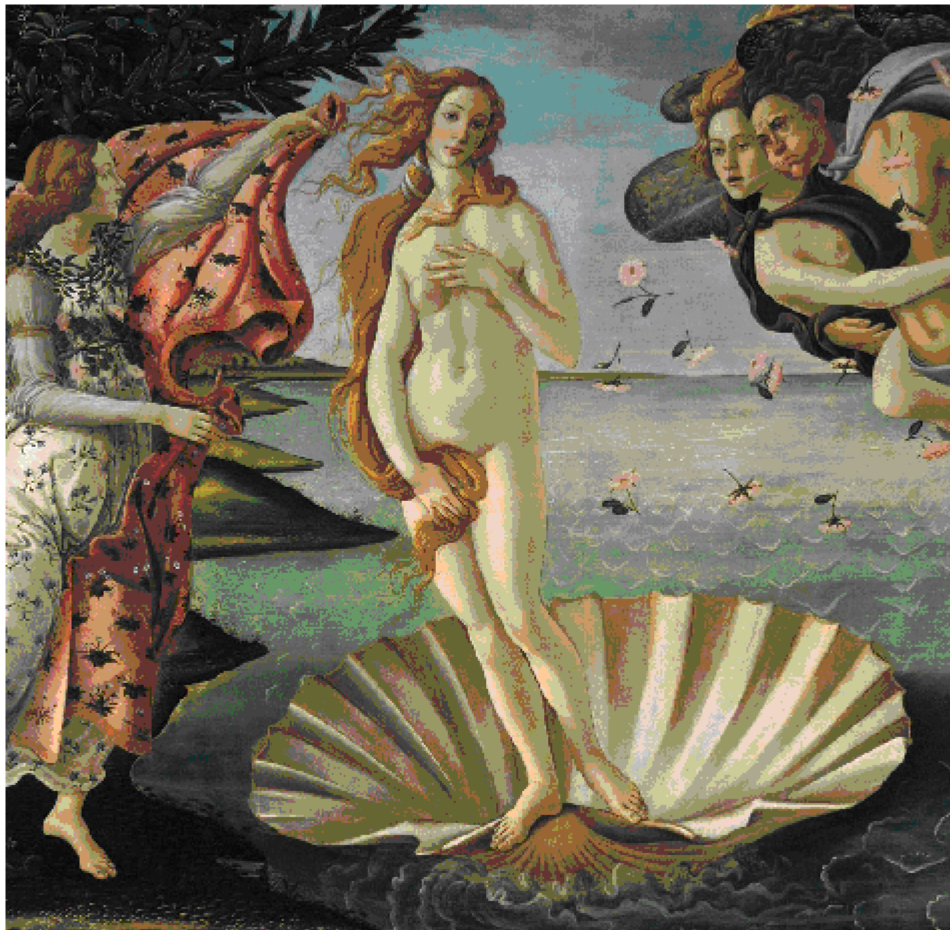
This document is excerpted from AztecC_minimanual.txt which was produced by Rubywand (Jeff Hurlburt). Despite the fact that I plagiarize Jeff's stuff shamelessly I can take no credit for his large body of work on behalf of the Apple II community.

In fact without Jeff's work in documenting the Apple II and the contribution of his cohorts from csa2 I could take little credit for my own work on the subject.

I have been formatting Rubywand's Aztec C Minimanual as part of a larger project. The Tutorial Introduction and Shell Sections (this document) are done so I am distributing those excerpts in advance of the whole manual.

All the Best,

Bill Buckels
bbuckels@escape.ca
August 2013



2. THE SHELL

2.5 General Use

The simplest form of a SHELL command is the name of a function followed by a carriage return. A SHELL command may either be one of the built-in utilities or the name of a binary or text file which resides on disk. The following is a list of the built-in functions available with the SHELL. If a file has the same name as one of these functions, the SHELL will not execute that file, but will execute the built-in function instead.

boot	**cp**	**mv**
bye	**load**	**rm**
call	**lock**	**run**
cat	**ls**	**save**
cd/ce	**max files**	**unlock**

These commands are all specified using lower case. A complete description of each command can be found in the Commands section.

Binary programs which are normally run using the DOS 'BRUN' command can be loaded and executed by simply typing the name of the file followed by a carriage return. The first two words of binary files which contain executable programs contain the load address and length in bytes of the memory image. These are used to load the program into memory. The SHELL 'load' command can be used to load the image into a different section of memory much the same as the DOS 'BLOAD' command.

Text files containing a series of SHELL command lines can be executed by simply typing the name of the text file followed by a carriage return. All SHELL input is then taken from that file until the end is reached For more information see the Batch Facilities section.

Some built-in SHELL utilities as well as binary programs produced using the Aztec C compiler system require or allow arguments to be specified when the command is executed These arguments are placed on the same line as the command name separated by spaces. An example of this is the SHELL 'lock' command Under Apple DOS, if it is desired to lock several files, the DOS 'LOCK' command must be given once for each file. To lock several files using the SHELL, the user would type something like:

lock test1 test2 test3,d2

to lock files "test1" and "test2" on the current drive and "test3" on drive two.

Because arguments are separated by spaces, file names containing spaces must be enclosed in double quotes to enable the SHELL to distinguish the single name from two names. For example, to unlock a file called "testprog", the user would type:

unlock "testprog"

Double quotes should also be used around file names used as commands if the name contains any blanks.

The final feature of the SHELL which will be discussed is the ability to redirect the standard input and/or output of a program to a file or a device. Normally the standard input and output of a program are connected to the keyboard and screen respectively. The user may redirect either or both of these connections to a file or a device such as a printer. This is accomplished by using the special character '<' for input and '>' for output.

As an example, to place the output of the NM command, which produces a symbol table from an object file, into a file for later perusal, type:

nm objfile > listing

The namelist will not be printed to the screen, but to the file 'listing' instead. The SHELL also pre-opens a second channel to the screen
called the standard error output. This channel cannot be redirected.

2.6 Built-in Commands

This section describes the commands which are built into the SHELL program itself. Each SHELL command will be listed along with a description of
its use and its function. All commands are specified as being lower case. File names may be typed with either upper or lower case letters, however they will all be mapped to upper case for compatibility with Apple DOS. File names may contain blanks, but to distinguish arguments from the parts of the file name, the entire name must be enclosed within double quotes.

In the following discussions, the concept of current data drive and current execution drive are used Under DOS, the last drive accessed is considered the current drive. Under the SHELL, the current data slot, drive and volume must be explicitly changed by the user using the "cd" command at any point where an optional slot, drive or volume parameter may be given. If any are not specified, they will default to the current data value respectively. The examples given for specific commands should clarify this point.

In general, all arguments to SHELL commands and to utility programs are separated by blanks. Arguments in square brackets are optional and most commands allow more than one file name per command line. In the following descriptions, any reference to a file name is assumed to include the optional slot, drive and volume parameters.

2.6.1 boot

boot n

Does a jump to (slot number n) address $Cn00. (Usually to reboot a drive controller installed at slot n (4, 5, 6 or 7).)

Example:

boot 6

Causes the floppy disk to reboot

Note: The boot command is really the equivalent of typing "PR#" in a BASIC program; boot 6 will reboot a floppy drive system with the floppy controller in slot 6. If you have a Microsoft CP/M Softcard in Slot 4 or 5, or an Applicard in slots 4, 5, or 7, this command can be used to get to CP/M from DOS 3.3 by typing boot followed by the respective slot number. Since hard disk systems aren't really supported by the DOS 3.3 filing system, boot 7 for a hard drive would be unlikely.

Doing a boot 3 with an 80 column card installed which has the same effect as a jump to $C300 messes-up the shell's text screens and creates double spacing. Configuring the shell to 80 column mode and letting the shell take care of text screens is the only alternative, and a jsr or a jump to $C300 should never be done in a shell program if you want your text screens to work afterwards. The shell does its own thing when it comes to screen control and you are best to avoid any of the "RAW" jumps and jsrs to manipulate text screens in shell programs. The shell has routines for that, and the text screen cursor control and clearing the screen are supported by the shell's internal terminal routines. To recap, this command is useful to reboot your floppy (boot 6) including in a shell script that is not being redirected.

2.6.2 bye

bye

Does a jump to the Apple machine language monitor at location $FF65. Re-entry to the SHELL is through $3D0 or by hitting RESET on systems with the autostart ROM.

2.6.3 call

call addr

Performs a "jsr" to the address given. If addr is preceded by a '$', it is interpreted as hex, otherwise as decimal.

Examples:

call $800
call -151

The first example does a "jsr" to hex 800, while the second calls the monitor.

Note: Careful using this one. It probably works ok to run a little bit of code loaded into where the shell expects, like at $800. It mucks-up on calls to routines like catalog at $a56e. Equivalent shell commands can be used in some cases; i.e. ls maps to catalog and works properly. In your own shell programs, using the runtime library calls supplied with Aztec C like the catalog() call is a better alternative, but in a "RAW" DOS 3.3 program you are free to do what you want. However, those "RAW" programs that do what they want don't always run properly in the Aztec C65 DOS 3.3 shell. The newer ProDOS Shell is more forgiving.

2.6.4 cat

cat [file1] [file2] ...

Concatenates the named files to the standard output. If no files are specified, input is taken from the standard input. This is the quickest and easiest way of looking at a text file.

Examples:

cat test1 test2,d1
cat test1 test2,d1 > test3
cat > pr:
cat kb: > pr:
cat > myfile.txt

The first example displays "test1" from the current data drive on the screen immediately followed by the file "test2" located on drive one. The second example creates a new file called "test3" containing the two files "test1" and "test2". The third example reads a character from the standard input and writes it to the device "pr:" which is the printer. The fourth example is equivalent to the third.

Note: One of the most useful variations of the cat command was never included in the orginal manual. I have placed this as example 5 in the examples above. By redirecting to a text file and then pressing ctrl-c (ctrl-break) when done, you can create shell scripts and other 7 bit sequential ascii files without the need of an editor. The gotcha' here is that you can't miss a typo on a previous line or insert a line above the current line, because all you are doing is copying stdin to stdout. You can have blank lines in these files and pretty much any character that the Shell console driver accepts.

2.6.5 cd

cd sn,dn,vn

Change the current data slot, drive and/or volume. Any or all of the three parameters may be changed. Those not specified will remain the same. If a volume number is specified., it will be checked whenever a file is opened A volume number of zero, however, will match any disk.

Examples:

cd s6,d1,v0
cd d2

The first example changes the current data disk to be slot six, drive one, and any volume. The second example changes from whatever the current drive was to drive two. The slot and volume remain the same.

2.6.6 ce

ce sn,dn,vn

Change the current execution slot, drive and/or volume. Execution parameters are used when loading and running a particular binary program or SHELL file. If the name includes a specific reference to a slot, drive or volume, that parameter is used If there is no reference as to which device holds the file, the current data disk is searched and if the file is not found there, then the current execution disk is checked This allows all utility programs to reside on a different disk than the one being actively used.

ce s6,d2,v0
ce d1

The first example changes the current execution disk to be slot six, drive two, and any volume. The second example changes from whatever the current drive was to drive one. The slot and volume remain the same.

2.6.7 cp

cp file1 file2

Copies files from the specified device to file2. Note that file2 will be overwritten if it already exists.

Examples:

cp test oldtest

cp test,d1 test

The first example makes a copy of "test" on the same disk called "oldtest". The second example assumes that drive one is not the current data drive and copies the file "test" from drive one to the current drive.

2.6.8 load

load file [aN] [lN]

Loads a binary file into memory. If the starting address and/or length are not specified, they are taken from the first two words of the file. After loading, the start address and length are displayed on the screen. These values are remembered for use in the save and run commands. If N begins with a '$', the value is interpreted as a hex value otherwise as decimal.

Examples:

load tabset
A=0800 L=12F2 (read from the binary file header)

load tabset a$2000
A=2000 L=12F2 (length read from the binary file header)

The first example loads the tabset program into memory. The shell displays the load address and length. The second example loads the tabset program into memory at address hex 2000.

2.6.9 lock

lock file 1 [file2] ...

Lock the file on the specified slot, drive and volume. If any of slot, drive or volume are not given, they default to the current data values.

Examples:

lock test1
lock test1 test2 test3,d2

The first example locks file test1 on the current data disk. Example two locks files test1 and test2 on the current data disk and locks file test3 on drive two of the current data slot and volume.

2.6.10 ls

ls [sn,dn,vn] ...

Perform the catalog function on the specified slot, drive and volume. This command defaults to the current data slot, drive and volume. If more than one is specified, they will be cataloged in order. The SHELL will wait for a key to be pressed between different catalogs. Unfortunately, the output of ls cannot be redirected.

Note: 3 example programs provided with the Aztec33 distribution provide different methods of creating text files of directory lists. Two of these are for the shell; one (called DLIST) uses the C65 runtime catalog() function and the other (called LS33) uses a lower level C65 function called rwts() (Read Write Track Sector), which is also in the Aztec C65 runtime library. Output from LS33 can be redirected and options are provided for search criteria based on file type or extension or both. LS33 also creates shell command scripts as an output option. The other two programs (DIR33 and DLIST) write text files. LS33 and DIR33 (its "RAW DOS 3.3" equivalent) both provide output in 7 bit plain text or DOS 3.3 text. But the Shell's ls command is simply a wrapper for their runtime catalog() function, and although necessary it cannot be redirected to create lists and it cannot be scoped to provide a selective listing:

Examples:

ls
ls d1 d2

The first example does a catalog of the current data slot, drive and volume. The second example catalogs drive one and then drive two of the current data slot and volume.

2.6.11 maxfiles

maxfiles n

Allocates n buffers for open files. This command is similar to the DOS 'MAXFILES' command. It specifies the maximum number of disk files which may be open at anyone time. When the SHELL is initialized, the value is defaulted to 3.

Example:

maxfiles 4

For an application which will have four disk files open, maxfiles is set to four.

2.6.12 mv

mv [-f] file1 file2

Moves file1 to file2. If the slot, drive, and volume of file1 are the same as that of file2, file1 is simply renamed as file2. If they are different, file1 is copied to file2 on the specified device and file1 is deleted. If file2 exists, an error message will be printed. If the '-f' option is given, no error message will be given and file2 will be removed first.

Examples:

mv test foo
mv -f test foo
mv test test,d2

The first example simply renames the file "test" as "foo". The second example deletes the file "foo" and then renames "test". The last example copies the file "test" from the current data drive to drive two and then deletes "test" from the current drive.

2.6.13 rm

rm file1 [file2] ...

Delete the specified file or files. If a file is locked, a message is displayed giving the name of the file which is locked.

Examples:

rm file1 file2
rm foo,s5

The first example deletes files "file1" and "file2" from the current data drive. The second example deletes the file "foo" from the disk in slot five. The drive number will be the same as the current data drive number.

2.6.14 run

run [argl] [arg2] ...

Does a jsr to the starting address of the last file loaded after pushing a pointer to the argument vector and the number of arguments on the stack. Argv[0] will be the "run" string.

Example:

load tabset
A=0800 L=12F0
run 8

This example loads the program "tabset" into memory. The SHELL displays the load address and length. The "run" command then calls hex 800 ($800) with the argument "8". The three lines are equivalent to typing:

tabset 8

all by itself.

2.6.15 save

save file [aN] [lN]

Saves a part of memory to a file on the specified device. If the starting address and length are not specified, the starting address and length of the last file "load"ed will be used if N is begun with a '$', the value is interpreted as a hex value otherwise as decimal.

Examples:

save foo
save foo a$800 l1000

The first example will save in a file called "foo", whatever the last program loaded or run. The second example will save a thousand bytes of memory starting at hex 800 in a file called "foo".

2.6.16 unlock

unlock file1 [file2] ...

Unlock the file on the specified slot, drive and volume. If any of slot, drive or volume are not given, they default to the current data values.

Examples:

unlock test1
unlock test1 test2 test3,d2

The first example unlocks file test1 on the current data disk. Example two unlocks files test1 and test2 on the current data disk and unlocks file test3 on drive two of the current data slot and volume.

2.7 Batch Facilities

Text files (7 bit sequential text files not DOS 3.3 text files with hi-bits set) containing a series of SHELL command lines can be executed by simply typing the name of the text file followed by a carriage return. Note that the type of the file must be 'T'. All SHELL

input is then taken from that file until the end is reached SHELL command files may not be nested, but they may be chained. If a SHELL command line executes a second SHELL command file, the first command file is closed and forgotten. Lines beginning with the '#' character are ignored by the shell and can be used as comments.

When the SHELL is booted for the first time, the disk that the SHELL was booted from is searched for a file called ".PROFILE". If this file is found and is a (7 bit not DOS 3.3) text file, it will be executed immediately. This allow any special startup procedures to be automatically initiated. SHELL command files may also be given up to 9 arguments. These arguments are referenced by the character '$' followed by the number of the argument to be used. Argument 0 is the name of the SHELL command file itself. For example, to link together several files, the following one line SHELL command file might be created:

ln -o In.out $1 $2 $3 $4 $5 $6 $7 $8 $9 sh65.lib

If the file was called "linkit", it could be used by typing:

linkit f1.rel f2.rel f3.rel

If an argument does not exist, it is ignored.

There are two special "built-in" commands that the SHELL will only recognize when read from a SHELL command file. These commands are used for additional control over the processing of the commands in a SHELL command file.

2.7.1 loop

loop

This command is used to start and end a loop in a SHELL command file. The command lines between the two loop statements will be executed once for each argument given to the SHELL command file. During the loop, two special arguments are available for use.

'$#' is replaced by the number of the current argument being processed. The two-character sequence '$%' will be replaced by the current argument itself. The following is an example of a SHELL command file which will compile and assemble from one to nine files, one at a time.

```
set -x -a
loop
# This is argument number $#, $%
c65 -a -o $%.asm $%.c
as65 -o $%.rel $%.asm
loop
```

If the preceding lines were placed in a file called "compile", then the statement:

compile test junk foo

would compile and assemble the three files "test.c", "junk.c", and "foo.c" into the corresponding ".rel" files and produce:

loop
This is argument 1, test
c65 -a -o test.asm test.c
as65 -o test.rel test.asm
loop
This is argument 2, junk
c65 -a -o junk.asm junk.c
as65 -o junk.rel junk.asm
loop
This is argument 3, foo
c65 -a -o foo.asm foo.c
as65 -o foo.rel foo.asm
loop

2.7.2 set

set [+-x] [+-a] [+-n]

Sets or clears one of three internal flags in the SHELL. Using '+' will clear the flag while '-' will set it. The flags are defined as follows:

x	**Echo command lines to the screen. Defaults to off.**
a	**Abort the SHELL command file if a command or program exits with a non-zero value. Defaults to no abort.**
n	**Parse the command lines but do not execute them. Defaults to off.**

Thus, to see each line being executed, the first line of a SHELL command file should be:

set -x

To have a SHELL command file exit if an error occurs, include the line:

set -a

The "set" command may only be executed within a SHELL command file.

2.8 Configuration

The basic Apple II is limited in its ability to deal with upper and lower case and has a limited screen size. The SHELL contains device drivers which allow it to overcome these limitations to some degree. However, these same routines have been set up to take advantage of optional peripherals which greatly enhance the Apple's operation. There are two approaches to dealing with peripheral devices, writing custom routines to deal with one particular device or to write a general routine to handle a number of similar devices. The original versions of the SHELL device drivers were examples of writing custom routines. The current version contains general purpose routines for dealing with three devices, the keyboard, screen and printer.

The device routines make use of a table at a fixed location in the driver to handle the functional differences between different hardware configurations. This table can be modified by using the CONFIG program provided on the STARTUP diskette. A separate set of options is available for each device and are detailed in the following.

2.8.1 Keyboard

The first device is the keyboard There are four variations of keyboard available. First, is a full upper and lower case keyboard as is available with the //e or a keyboard enhancer. If this option is selected, no mapping is done at all. Second, is an Apple keyboard with the single wire shift key mod installed, while the third is an Apple keyboard without the SWSKM. Both of these options map characters from the keyboard to get the full range of ASCII characters. Finally, it is possible to specify that the keyboard is a remote terminal. In this case, the driver will use the Pascal 1.0 entry point to the card that is assumed to be in slot 3. It will not do any mapping on the data received from the card. Also, since there is no status entry point, the AS and AC output control characters are not available. The AC abort is still enabled during input.

2.8.2 Screen

The second device is the screen. There are three types of screen. First, the basic Apple screen with 40 columns and upper case only. Second, 40 columns with upper and lower case capability. Examples of this are the //e and a II with a lower case adapter. Finally, there are the 80-column screens. All 80-column screens, remote and otherwise, are assumed to reside in slot 3 and are accessed by using the Pascal 1.0 output hook.

Not all 80-column screens are identical, and do not necessarily use the same control sequences to perform such functions as clearing the screen, moving the cursor and others. To minimize this problem, a table of control codes has been built into the SHELL device driver. This table is used by the ioctl() routine when performing the appropriate functions.

The values in this table are already known for several devices by the CONFIG program. The devices whose values are known are the //e, the Videx Videoterm, and the Smarterm. If a device is used which is not compatible with any of the above three cards, then the entries to the table must be provided by the user. The only programs which currently

make use of the ioctl() screen calls are the screen editor VED, and the CONFIG program. For VED, the only required functions are cursor positioning, clear to end of line, and clear screen. CONFIG only uses the clear screen.

2.8.3 Printer

The last device is the printer. The printer is assumed to be driven by a peripheral card in slot 1 with firmware which supports the PR# basic protocol. The printer is initialized by placing the address of the card in the CSW vector in low memory and then calling the card. Normally the card then replaces the address in the CSW vector with the normal character output routine. The printer driver then sends a string of characters to initialize the firmware on the card. The default sequence is:

^I^Y^Y255N

which tells the card that the width is 255 and not to echo the characters to the Apple screen. It is not really necessary to change the control character to be something other than a ^I since tabs are expanded to spaces by the print driver. After the driver sends the initialization string, it saves the address in CSW for use when sending characters to the printer. When sending characters, the address is placed back in CSW and control passed to the firmware by jumping indirectly through CSW.

The printer has three other control modes. First, some printer card firmware requires that the high bit be on for characters sent to it. If so, the driver has a flag which will cause it to "or" in a hex 80 with each character before transmitting it to the firmware. Also, the print driver automatically converts newlines (LF) to carriage returns before transmitting to the firmware on the card. If the appropriate flag is set, the print driver will automatically send a line feed after each carriage return. Finally, when the printer is closed it is possible to have the print driver automatically send a form feed ($0C) character to the device. All these flags are set by answering the appropriate questions in the CONFIG program.