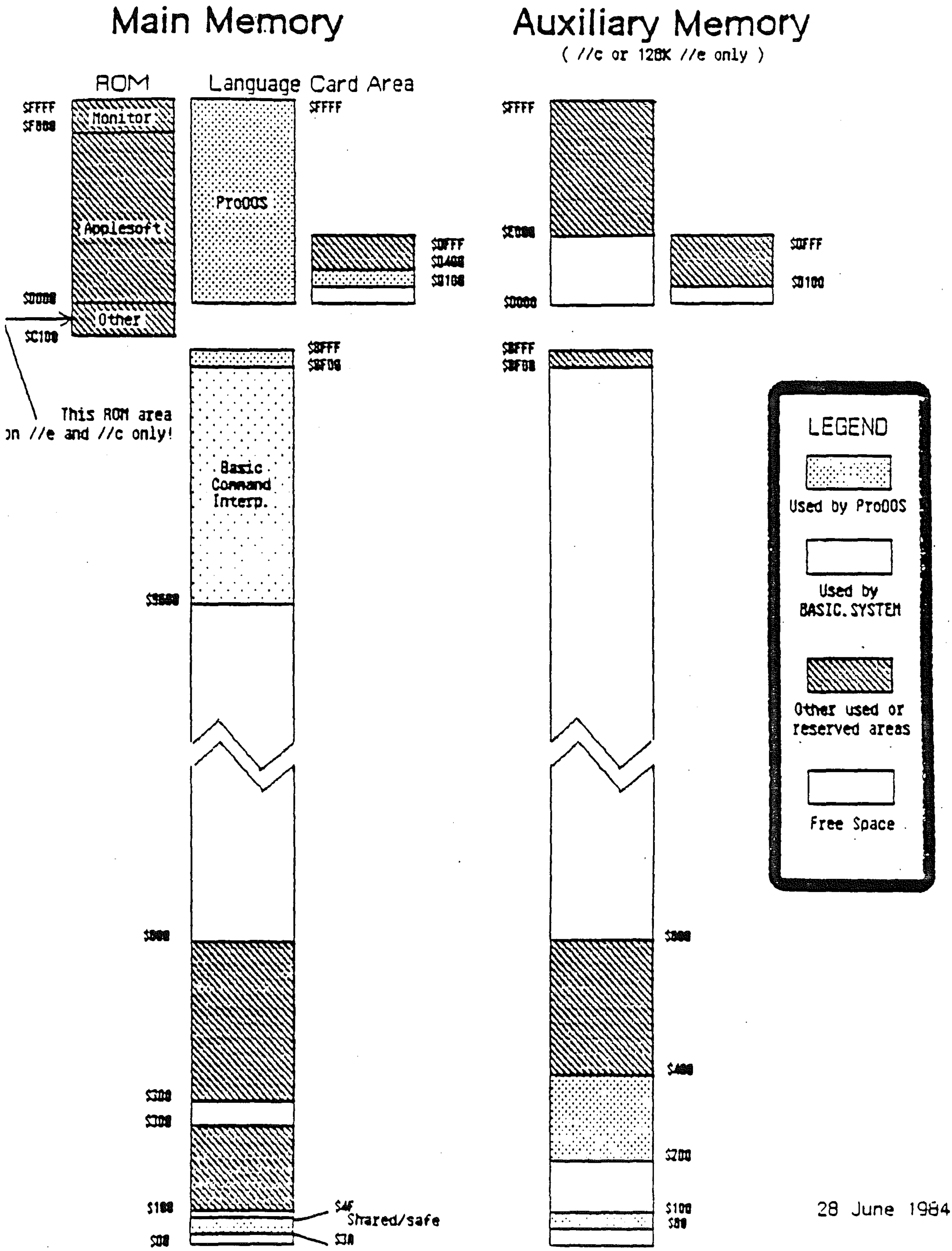
ProDOS Technical Notes

Revised May 08, 1984

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

ProDOS Memory Map



28 June 1984

ProDOS TECHNICAL NOTE #1

The GETLN Input Buffer and the ThunderClock

(14 July 1983)

The ThunderClock is automatically supported by ProDOS when ever it is identified as installed in the system. When programming under ProDOS, always consider the ThunderClock's impact on the GETLN input buffer ($200 - $2FF). ProDOS can support other clocks which may also use this space.

When ever the ThunderClock receives a call from ProDOS, it deposits an ASCII string in the GETLN input buffer of the form:

07,04,14,22,46,57

which translates as:

07 = The month, JULY (01=JAN,...,12=DEC)
04 = The day-of-the-week, THURSDAY (00=SUN,...,06=SAT)
14 = The date, 14th (00 to 31)
22 = The hour, 10PM (00 to 23)
46 = The minute (00 to 59)
57 = The second (00 to 59)

ProDOS calls the ThunderClock as part of many of its routines. Anything in the first 17 bytes of the GETLN input buffer is subject to loss if a ThunderClock is installed and gets called.

It has been the practice of programmers, in general, to use the GETLN input buffer for every conceivable purpose. Therefore, an application should never store anything there. If your application has future need to know about the contents of the $200-$2FF space, it should be transferred to some other location to guarantee it will remain intact, particularly under ProDOS where a ThunderClock may regularly be overwriting the first 17 bytes.

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #2

Notes on Transporting DOS Assembly Language Programs to ProDOS
(Passing Disk Commands Under BASIC.SYSTEM to ProDOS from Machine Code.)

(Revised August 7, 1984)

Under DOS, commands were executed by a direct call to the proper address in DOS or by sending a string to COUT ($FDED) consisting of [CTRL-D] <command> [RETURN].

The practice that became very common under DOS of making direct calls to the desired routines within DOS cannot be carried over to ProDOS. Apple Computer will not support any entries into the BASIC Command Interpreter or the ProDOS kernel that are not published by Apple. If you use any undocumented entries, your application will almost certainly not operate under future releases of PRODOS and BASIC.SYSTEM.

Passing disk commands as ASCII strings to COUT is not supported under ProDOS.

If you wish to issue a ProDOS command from a machine language module operating with Applesoft or if your application can permit the ProDOS BASIC Command Interpreter (BASIC.SYSTEM) to be co-resident in memory, you can still use an ASCII string. All that is necessary is to move the string, ending with a RETURN ($8D) to the GETLN buffer ($200) and execute a JSR DOSCMD ($BE03) to execute the instruction at $200.

*** It is necessary that the JSR DOSCMD be performed in deferred mode (inside a program) and not in immediate mode. This also applies to the monitor program; while in the monitor you cannot do a $xxxxG to execute the code that contains the JSR DOSCMD. The reason for this is that BASIC.SYSTEM checks certain state flags. These flags are set correctly for the DOSCMD routine only while in deferred mode. DOSCMD was intended only to be used via a CALL inside a BASIC program.

There are certain commands that will NOT work correctly or as expected when initiated via DOSCMD. The following table lists those commands which work properly and those that do not.

PLEASE NOTE that some of the commands listed as not working properly may work well enough to suit your individual purposes. Also some commands will function (albeit precariously) in immediate mode.
IF YOU DECIDE TO USE THE COMMANDS IN THIS MANNER YOU ARE ON YOUR OWN.

Attached is an example BASIC program that will BLOAD an assembly routine that will exercise the DOSCMD routine. The BASIC program is first LISTed and then RUN. A listing of the assembly routine follows. Please review it before writing your own routine.

DOSCMD is merely a means of performing some BASIC.SYSTFM commands from assembly language and is not a substitute for performing the commands in BASIC. Keep in mind all the conseqences of the command you are executing; EG. When doing a BRUN or BLOAD make sure the code is loaded at suitable addresses.

Error Handling

Right after you call DOSCMD the carry bit will tell you whether or not an error had occurred. The carry will be set if an error had occurred. The accumulator will always have the error number.

DOSCMD error handling can be handled in one of three ways:

1. Do a JSR ERROUT ($BE09). This will return control to your BASIC ONERR routine where you can then handle the error.

2. Do a JSR PRINTERR ($BE0C). This will print out the error and will return control to the point after the JSR (as usual).

3. You can handle the error yourself completely. If choose to go this route make sure you clear the carry (CLC) before you return control back to BASIC.SYSTEM. If you don't it will be assumed some error has occurred and will do awful and unpredictable things to you.

Works Correctly and Returns Control to Calling Routine	Works Incorrectly and/or does not Return Control to Calling Routine
Filing Commands:	
Catalog, Cat	
Prefix, Prefix /pn	
Create	
Rename	
Delete	
Lock	
Unlock	
Program Commands:	
	- (Dash)
	Run
	Load
Save	
Programming Commands:	
	Chain
Store	
Restore	
Pr#	
In#	
Fre	
Text File Commands:	
Open	
Close	
	Read
	Write
	Append
Flush	
Position	
EXEC Command:	
	Exec
Binary Commands:	
Brun	
Bload	
Bsave	

```
10   REM  YOU MUST CALL THE ROUTINE FROM INSIDE A BASIC PROGRAM!!
     REM
12   REM
20   PRINT  CHR$ (4)"BLOAD/P/PROGRAMS/CMD.0"
30   CALL 4096
40   PRINT "BACK TO THE WONDERFUL WORLD OF BASIC!"
50   END

]RUN


ENTER BASIC.SYSTEM COMMAND --> PREFIX

/P/


ENTER BASIC.SYSTEM COMMAND --> PREFIX/P/BUGS

ENTER BASIC.SYSTEM COMMAND --> PREFIX

/P/BUGS/


ENTER BASIC.SYSTEM COMMAND --> CATALOG

BUGS

 NAME             TYPE  BLOCKS  MODIFIED          CREATED           ENDFILE SUBTYPE

*SEQTEST           DIR       1  23-APR-84 16:12   23-APR-84 16:12       512
 WRITEFIELDS       BAS       1  27-MAR-84 15:00   23-APR-84 16:13       182
 R                 BAS       1  27-MAR-84 15:29   23-APR-84 16:13       193
 READFIELDS        BAS       1  27-MAR-84 15:17   23-APR-84 16:13       185
 DUMPFILE          BAS       1  27-MAR-84 11:01   23-APR-84 16:13       191
 POSTEST           BAS       1  27-MAR-84 16:50   23-APR-84 16:13       174
 MAKEJUNK          BAS       1  29-MAR-84 14:10   23-APR-84 16:14        82
 P1                BAS       1   3-AUG-84 17:53   23-APR-84 16:15       416

BLOCKS FREE: 6215      BLOCKS USED: 3513      TOTAL BLOCKS: 9728


ENTER BASIC.SYSTEM COMMAND --> DO DA, DO DA

SYNTAX ERROR
BACK TO THE WONDERFUL WORLD OF BASIC!
```

```
SOURCE   FILE #01 =>/P/PROGRAMS/CMD
----- NEXT OBJECT FILE NAME IS /P/PROGRAMS/CMD.0
1000:          1000                 ORG    $1000
1000:          FD6F    GETLN1       EQU    $FD6F         ; MONITORS INPUT ROUTINE
1000:          BE03    DOSCMD       EQU    $BE03         ; BASIC.SYSTEMS GLBL PG DOS CMD ENTRY
1000:          FDED    COUT         EQU    $FDED         ; MONITORS CHAR OUT ROUTINE
1000:          BE0C    PRERR        EQU    $BE0C         ; PRINT THE ERROR
1000:                  *
1000:                  *
1000:                  *
1000:A2 00             START        LDX    #0            ; DISPLAY PROMPT...
1002:BD 1F 10          L1           LDA    PROMPT,X      ;
1005:F0 06     100D                 BEQ    CONT          ; BRANCH IF END OF STRING
1007:20 ED FD                       JSR    COUT          ;
100A:E8                             INX                   ;
100B:D0 F5     1002                 BNE    L1            ; LOOP UNTIL NULL TERMINATOR IS HIT...
100D:                  *
100D:20 6F FD          CONT         JSR    GETLN1        ; NOW ACCEPT USER COMMAND FROM KB
1010:20 03 BE                       JSR    DOSCMD        ; AND EXECUTE THE COMMAND
1013:2C 10 C0                       BIT    $C010         ; CLEAR STROBE SO KEY WON'T HANG AROUND..
1016:B0 02     101A                 BCS    ERROR         ; BRANCH IF ERROR DETECTED
1018:90 E6     1000                 BCC    START         ; OTHERWISE RESTART....
101A:                  *
101A:                  *
101A:                  * NOTE: AFTER HANDLING YOUR ERROR YOU MUST CLEAR THE CARRY
101A:                  *       BEFORE RETURNING TO BASIC OR ELSE BASIC WILL DO
101A:                  *       STRANGE THINGS TO YOU.
101A:                  *
101A:20 0C BE          ERROR        JSR    PRERR         ; PRINT 'ERR'
101D:18                             CLC                   ;
101E:60                             RTS                   ; RETURN TO BASIC
101F:                  *
101F:                               MSB    ON
101F:                  *
101F:8D                PROMPT       DB     $8D           ; OUTPUT A RETURN FIRST
1020:C5 CE D4 C5                    ASC    'ENTER        BASIC.SYSTEM COMMAND --> '
103F:00                             DB     0
```

```
100D CONT           FDED COUT           BE83 DOSCMD          101A ERROR
FD6F GETLN1         1002 L1             BE8C PRERR           101F PROMPT
1    START
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED     35
** FREE SPACE PAGE COUNT    89
```

ProDOS TECHNICAL NOTE #3

ProDOS Device Search and Identification Procedure
Disk Driver Conventions

(Revised 20 December 1983)

During boot-up, ProDOS does a device search looking for block storage devices. As described in the ProDOS Technical Reference Manual, all disk drives must "look and act just like one of our drives".

ProDOS looks for the following:

$CnO1 = $20 $Cn03=$00 $Cn05=$03

where n = the slot number. Having found these three bytes in the ROM of a particular slot, ProDOS assumes it has found a disk drive.

If $CnFF=$00 ProDOS assumes it has found a Disk][with 16-sector ROMs and marks the device driver table in the ProDOS global page with the address of the Disk][driver routines. The Disk][driver routines will support any drive that "looks and acts like a Disk][" (280 blocks, single volume, etc.).

If $CnFF=$FF, ProDOS assumes it has found a Disk][with 13-sector ROMs and makes no attempt to support the device 13-sector ROMs since it may not operate properly under ProDOS.

If ProDOS finds a value other than $00 or $FF at $CnFF, it assumes it has found an "intelligent" disk controller. If the STATUS BYTE at $CnFE indicates that the device supports READ and STATUS requests, ProDOS marks the global page with a device driver address whose high-byte is equal to $Cn and whose low-byte is equal to the value found at $CnFF. Intelligent controller cards CANNOT be auto-bootable due to a conflict with Pascal which believes all auto-boot devices are Disk][floppy drives. (Therefore, the byte at $Cn07 must not be $3C.)

The only calls to the disk driver are STATUS, READ, WRITE, and FORMAT. The STATUS call should perform a check to verify that the device is ready for a READ or WRITE. If it is not, the carry should be set and the appropriate error code returned in the accumulator. If the device is ready for a READ or WRITE, then the driver should clear the carry, place a zero in the accumulator, and return the number of blocks on the device in the X-register (lo-byte) and Y-register (hi-byte).

If you wish to interface a disk controller card with more than two drives (or a device with more than two volumes), additional device driver vectors for disk controllers plugged into slot 5 or 6 may be installed in slot 1 or 2 locations. There will be no conflict with character devices physically present in these slots. Device numbers for four drives in slot five or slot six are listed below.

Physical	S5,D1 = $50	Physical	S6,D1 = $60
Slot	S5,D2 = $D0	Slot	S6,D2 = $E0
Five	S1,D1 = $10	Six	S2,D1 = $20
	S1,D2 = $90		S2,D2 = $A0

The special locations in the ROM code are:

$CnFC-$CnFD = The total number of blocks on the device. Used for writing the disk's bit-map and directory header after formatting. (If this location is $0000, it indicates that the number of blocks must be obtained by making a STATUS request.)

$CnFE = The status byte (bit 0 and 1 must be set for ProDOS to install the driver vector!)

```
Bit  7    - Medium is removable
Bit  6    - Device is interruptable
Bit  5-4  - Number of volumes on the device (0-3)
Bit  3    - The device supports formatting
Bit  2    - The device can be written to
Bit  1    - The device can be read from     (Must be on)
Bit  0    - The device's status can be read (Must be on)
```

$CnFF = The lo-byte of entry to the driver routines...ProDOS will place $Cn + this byte in the global page.

The locations where the call parameters are passed to the driver are:

$42 - COMMAND: 0 = STATUS request 1 = READ request
2 = WRITE request 3 = FORMAT request

NOTE: The FORMAT code in the driver need only lay down address marks if required...the calling routine should write the "virgin directory and bit-map".

$43 - UNIT NUMBER:

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
|,DR|  ,SLOT    |   not  used   |
+---+---+---+---+---+---+---+---+
```

NOTE: The UNIT_NUMBER that appears in the device list (DEVLST) in the system globals will include the hi-nybble of the status byte ($CnFE) as an I.D. in it's lo-nybble.

$44-$45 - BUFFER POINTER: Indicates the start of a 512-byte memory buffer for data transfer.

$46-$47 - BLOCK NUMBER: Indicates the block on the disk for data transfer.

The device driver should report errors by setting the carry flag and loading the error code into the accumulator. The error codes that should be implemented are:

$27 - I/O error $28 - No device connected $2B - Write Protected

ProDOS TECHNICAL NOTE #4

Notes on Transporting DOS Assembly Language Programs to ProDOS (Redirecting I/O and converting "JSR $3EA")

(26 July 1983)

When programming under DOS 3.3, if you wished to change the I/O hooks, all that was necessary was to install your I/O routine addresses in the character-out vector ($36-$37) and/or key-in vector ($38-$39) and notify DOS (JSR $3EA) to take your addresses and swap it's intercept routine addresses in.

Under ProDOS, there is no instruction installed at $3EA at all. So what do you do?

Just leave the ProDOS Basic Command Interpreter's intercept addresses installed in $36-$39 and install your I/O addresses in the global page at $BE30-$BE33. $BE30-$BE31 should contain the output address (normally $FDF0, the monitor COUT1 routine), and $BE32-$BE33 should contain the input address (normally $FD1B, the monitor KEYIN routine).

By keeping these vectors in a global page, a special routine for moving the vectors is no longer needed, thus, no $3EA instruction. Just install the addresses at their destination yourself.

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #5

ProDOS Disk Formatting Routines

(11 January 1984)

The ProDOS Disk][FORMATTER and ProDOS BUILDDISK Routines are supplied as text files of source code. They can be assembled with the ProDOS version of EDASM, Apple's editor/assembler.

The source code for the FORMATTER was prepared with no labels so that you can "INCLUDE" it with your application at assembly time. Since disk I/O core routines MUST include critical, time dependent code, the FORMATTER source file MUST be assembled with the "ORG" on a page boundary. (Many instruction times change when page boundaries are crossed.)

The formatter routine uses zero page locations $DO thru $DD. If your application also uses these locations, you must save the contents prior to calling the formatter and restore them upon return.

When the routine is called, the ProDOS device number (DEVNUM) must be in the accumulator. DEVNUM in this case is defined as containing zeros in the low nibble, the slot number in bits 4, 5, 6, and the hi-bit set to zero for drive 1 or set to 1 for drive 2. Upon exit, if the carry flag is clear, no error has been detected and the accumulator will be zeroed.

If an error has been detected, the routine will exit with the carry flag set and the accumulator will hold the error code. Error codes that may be returned are: $27-unable to format, $28-write protected, $33-drive too slow, $34-drive too fast.

The FORMATTER routine ONLY writes zeros to each sector on a Disk][floppy. To install boot code, a directory and bit map, on any previously formatted storage device, you need the BUILDDISK routine.

Upon entry to the BUILDDISK routines the accumulator must contain the DEVNUM, X and Y must have the address of a 512 byte buffer (X-lo, Y-hi), and DUMMYNAM and DUMSIZE must be filled in with the desired volume name and name length if a name other than DEFAULT.NAME is desired.

BUILDDISK treats all devices the same, with two exceptions. These exceptions are identified by examining the low nibble of the DEVNUM. (Remember, the low nibble of the DEVNUM is derived from the high nibble of the device status byte at $CnFE in the ROM code.)

If all four bits of the i.d. nibble are set, BUILDDISK will assume that the device has unusual characteristics and that the driver has taken care of the bit map, directory and boot code during the format request. If all four bits are clear, BUILDDISK will recognize the device as a Disk][or Disk][emulator and assume the device has 280 blocks.

BUILDDISK leaves zero-page intact, with the exception of the bytes from $42 thru $47 which are defined for use when making requests to device drivers and standard ProDOS error codes will be returned.

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #6

Attaching External Commands to BASIC.SYSTEM

(Revised 19 September 1983)

Whenever BASIC.SYSTEM receives a command, it first checks it's command list, then sends it out to any external command handler and finally passes it on to Applesoft. If you find regular need for an additional command, you can write your own command handler and attach it to BASIC.SYSTEM through the EXTRNCMD jump vector. Just install the address of your routine in EXTRNCMD+1 and +2 (lo-byte first) and you're linked in. There are essentially three functions that your routine must perform.

(1) It must check for the presence of your command(s).

(2) If it is your command, it must let BASIC.SYSTEM know.

(3) It must execute the desired instructions expected of the command.

The first step (1) is quite straight forward, just inspect the GETLN input buffer. If it is not your command, a simple SEC and a RTS will return control to BASIC.SYSTEM to continue the search.

The second step (2) is more involved. It is your command, so you must zero XCNUM ($BE53) to indicate an external command and set XLEN ($BE52) equal to the length of your command string minus one.

If there are no associated parameters (such as slot, drive, A$, etc.) to parse, you must set all 16 parameter bits in PBITS ($BE54,$BE55) to zero. And, if you're going to handle everything yourself before returning control to BASIC.SYSTEM you must point XTRNADDR ($BE50, $BE51) at an RTS instruction...XRETURN ($BE9E) is a good location. Now just "fall through" to your execution routines (3).

If there are parameters to parse, it is easiest to let BASIC.SYSTEM parse them for you (unless you want to use some undefined parameters). By setting up the bits in PBITS ($BE54,$BE55), and setting XTRNADDR ($BE50,$BE51) equal to the location where execution of your command begins, you can return control to BASIC.SYSTEM, with an RTS, and let it parse and verify the parameters and return them to you in the global page.

The final step (3) is up to you and should RTS with the carry cleared.

Attached are two example routines, BEEP and BEEPSLOT. BEEP handles everything itself and BEEPSLOT will let you pass a slot & drive parameter (,S#,D#), where the drive is ignored.

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

BRUN BEEP.0 to install the routine's address in EXTRNCMD. Then type BEEP as immediate command or use PRINT CHR$(4);"BEEP" in a program.

```
0300:           0300            ORG   $300
0300:           0200   INBUF    EQU   $200          ;GETLN input buffer
0300:           FCA8   WAIT     EQU   $FCA8         ;Monitor wait routine
0300:           FF3A   BELL     EQU   $FF3A         ;Monitor bell routine
0300:           BE06   EXTRNCMD EQU   $BE06         ;External cmd JMP vector
0300:           BE50   XTRNADDR EQU   $BE50         ;Ex cmd implementation addr
0300:           BE52   XLEN     EQU   $BE52         ;Length of command string-1
0300:           BE53   XCNUM    EQU   $BE53         ;CI cmd no. (ex cmd = 0)
0300:           BE54   PBITS    EQU   $BE54         ;Command parameter bits
0300:           BE9E   XRETURN  EQU   $BE9E         ;Known RTS instruction
0300:                           MSB   ON            ;Set hi-bit on ASCII
0300:                  ;
0300:A9 0B                      LDA   #>BEEP        ;Install the address of our
0302:8D 07 BE                   STA   EXTRNCMD+1    ;  command handler in the
0305:A9 03                      LDA   #<BEEP        ;  external command JMP
0307:8D 08 BE                   STA   EXTRNCMD+2    ;  vector
030A:60                         RTS
030B:                  ;
030B:A2 00             BEEP     LDX   #0            ;Check for our command
030D:BD 00 02          NXTCHR   LDA   INBUF,X       ;Get first char
0310:DD 43 03                   CMP   CMD,X         ;Does it match?
0313:D0 2E      0343            BNE   RETURN        ;Nope, back to CI
0315:E8                         INX                 ;Next character
0316:E0 04                      CPX   #CMDLEN       ;All characters yet?
0318:D0 F3      030D            BNE   NXTCHR        ;No, read next one
031A:                  ;
031A:A9 03                      LDA   #CMDLEN-1     ;Our cmd! Put cmd length
031C:8D 52 BE                   STA   XLEN          ;  -1 in CI global XLEN
031F:A9 9E                      LDA   #>XRETURN     ;Point XTRNADDR to a known
0321:8D 50 BE                   STA   XTRNADDR      ;  RTS since we'll handle
0324:A9 BE                      LDA   #<XRETURN     ;  at the time we inter-
0326:8D 51 BE                   STA   XTRNADDR+1    ;  cept our command
0329:A9 00                      LDA   #0            ;Mark the cmd number as
032B:8D 53 BE                   STA   XCNUM         ;  zero (external)
032E:8D 54 BE                   STA   PBITS         ;And indicate no paramet
0331:8D 55 BE                   STA   PBITS+1       ;  to be parsed
0334:                  ;
0334:A2 05                      LDX   #5            ;Number of desired beeps
0336:20 3A FF          NXTBEEP  JSR   BELL          ;Else, beep once
0339:A9 80                      LDA   #$80          ;Set-up the delay
033B:20 A8 FC                   JSR   WAIT          ;  and wait
033E:CA                         DEX                 ;Decrement index and
033F:D0 F5      0336            BNE   NXTBEEP       ;  repeat til X = 0
0341:18                         CLC                 ;All done successfully
0342:60                         RTS
0343:                  ;
0343:38                RETURN   SEC                 ;Notify BASIC.SYSTEM it
0344:60                         RTS                 ;  it wasn't our command
0345:                  ;
0345:C2 C5 C5 D0       CMD      ASC   "BEEP"        ;Our command
0349:           0004   CMDLEN   EQU   *-CMD         ;Our Command length
```

BRUN BEEPSLOT.0 to install the routine's address in EXTRNCMD. Then enter BEEPSLOT,S(n),D(n). Only a legal slot and drive numbers are acceptable. If no slot number, it will use the default slot number. Any drive number is simply ignored. The command may also be used in a program PRINT CHR$(4) statement.

```
0300:         0300            ORG   $300
0300:         0200   INBUF    EQU   $200           ;GETLN input buffer
0300:         FCA8   WAIT     EQU   $FCA8          ;Monitor wait routine
0300:         FF3A   BELL     EQU   $FF3A          ;Monitor bell routine
0300:         BE06   EXTRNCMD EQU   $BE06          ;External cmd JMP vector
0300:         BE50   XTRNADDR EQU   $BE50          ;Ex cmd implementation addr
0300:         BE52   XLEN     EQU   $BE52          ;Length of command string-1
0300:         BE53   XCNUM    EQU   $BE53          ;CI cmd no. (ex cmd = 0)
0300:         BE54   PBITS    EQU   $BE54          ;Command parameter bits
0300:         BE61   VSLOT    EQU   $BE61          ;Verified slot parameter
0300:                         MSB   ON             ;Set hi-bit on ASCII
0300:                ;
0300:A9 0B                    LDA   #>BEEPSLOT     ;Install the address of our
0302:8D 07 BE                 STA   EXTRNCMD+1     ;  command handler in the
0305:A9 03                    LDA   #<BEEPSLOT     ;  external command JMP
0307:8D 08 BE                 STA   EXTRNCMD+2     ;  vector
030A:60                       RTS
030B:                ;
030B:A2 00           BEEPSLOT LDX   #0             ;Check for our command
030D:BD 00 02        NXTCHR   LDA   INBUF,X        ;Get first char
0310:DD 4B 03                 CMP   CMD,X          ;Does it match?
0313:D0 36    034B            BNE   RETURN         ;Nope, back to CI
0315:E8                       INX                  ;Next character
0316:E0 08                    CPX   #CMDLEN        ;All characters yet?
0318:D0 F3    030D            BNE   NXTCHR         ;No, read next one
031A:                ;
031A:A9 07                    LDA   #CMDLEN-1      ;Our cmd!  Put cmd length
031C:8D 52 BE                 STA   XLEN           ;  -1 in CI global XLEN
031F:A9 38                    LDA   #>EXECUTE      ;Point XTRNADDR to our
0321:8D 50 BE                 STA   XTRNADDR       ;  command execution
0324:A9 03                    LDA   #<EXECUTE      ;  routine
0326:8D 51 BE                 STA   XTRNADDR+1
0329:A9 00                    LDA   #0             ;Mark the cmd number as
032B:8D 53 BE                 STA   XCNUM          ;  zero (external)
032E:8D 54 BE                 STA   PBITS          ;And indicate that slot and
0331:A9 04                    LDA   #%00000100     ;  drive parameter may be
0333:8D 54 BE                 STA   PBITS          ;  accepted
0336:18                       CLC                  ;Everything if OK
0337:60                       RTS                  ;Return to BASIC.SYSTEM
0338:                ;
0338:AD 61 BE        EXECUTE  LDA   VSLOT          ;Get slot parameter
033B:29 0F                    AND   #%00001111     ;Zero the hi-bits
033D:AA                       TAX                  ;Transfer to index reg.
033E:20 3A FF        NXTBEEP  JSR   BELL           ;Else, beep once
0341:A9 80                    LDA   #$80           ;Set-up the delay
0343:20 A8 FC                 JSR   WAIT           ;  and wait
0346:CA                       DEX                  ;Decrement index and
0347:D0 F5    033E            BNE   NXTBEEP        ;  repeat til X = 0
0349:18                       CLC                  ;All done successfully
034A:60                       RTS
034B:                ;
034B:38              RETURN   SEC                  ;Notify BASIC.SYSTEM, it
034A:60                       RTS                  ;  wasn't our command
034B:                ;
034B:C2 C5 C5 D0     CMD      ASC   "BEEPSLOT"     ;Our command
0353:         0008   CMDLEN   EQU   *-CMD          ;Our Command length
```

ProDOS TECHNICAL NOTE #7

Starting and Quitting
Interpreter Conventions

(revised 09 March 1984)

It is absolutely essential that all interpreters (system programs) use a standard way of starting and quitting.

In order to provide a uniform method for starting and quitting, the following procedures are established and SUPERCEDE section 5.1.5 of the ProDOS Technical Reference Manual:

Starting:

System Programs are started by one of two ways:

1. The disk containing the ProDOS operating system and the system program is booted; ProDOS loads and runs the first XXX.SYSTEM file of type SYS($FF). The order of search is determined by the file entries in the boot volume directory.

2. The program is loaded by another program (like the ProDOS filer or the Basic Command Interpreter), or a program dispatcher (like the one that is part of ProDOS or a more sophisticated program selector).

The system program is loaded and jumped to at $2000. The complete or partial pathname of the system program is stored at $280 starting with a length byte. The string is a full pathname if it starts with a slash (/); it is a partial pathname if it starts with a letter.

The purpose of this pathname is to allow a system program to determine the directory where other needed files may reside. The program should NEVER assume that the files are in a specific directory or subdirectory.

Additionally, we establish a mechanism to pass a second pathname to interpreters which like to run STARTUP programs. An example of this is a language interpreter. The ProDOS dispatcher does not support this mechanism but other more sophisticated program selectors may.

The mechanism requires that the interpreter start a certain way:

- o $2000 is a jump instruction.
- o $2003 and $2004 are $EE.

If the interpreter starts this way, byte $2005 is assumed to be an indicator of the length of a buffer which starts at $2006 and holds the pathname (starting with a length byte) of the startup file.

Interpreters which support this mechanism should supply their own default string which should be a standard choice for a startup program or a flag not to run a startup program.

Once gaining control, the system program sets the reset vector and fixes the power-up byte. Never assume the state of the machine to be anything that is not clearly documented.

Note: If your interpreter makes use of the dispatcher/ selector area (addresses $D100-$D3FF in the second 4K-byte bank of RAM), be sure that this area is initially saved and then restored on exit.

Quitting:

1. Do normal housekeeping... close files, reinstall /RAM if you have had it disconnected, etc.

2. Trash the power-up byte at $3F4. The simplest way to do this is either to increment or decrement it, which will always make it an invalid check of the $3F2 vector.

3. Execute a ProDOS system call number $65 as follows:

```
EXIT       JSR  PRODOS       ; Call the MLI ($BF00)
           DFB  $65          ; CALL TYPE = QUIT
           DW   PARMTABLE    ; Pointer to parameter table

PARMTABLE  DFB  4            ; Number of parameters is 4
           DFB  0            ; 0 is the only quit type
           DW   0000         ; Pointer reserved for
;                              future use
           DFB  0            ; Byte reserved for future
;                              use
           DW   0000         ; Pointer reserved for
;                              future use.
```

It is most important to note that even though most of the parameter table is reserved for future use, it must all be present! It must consist of seven bytes... a $04 followed by six nulls ($00).

For more information on Dispatcher/Selector Conventions please see ProDOS Technical Note #14.

APPLE COMPUTER INC., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone: (408) 996-1010

ProDOS Technical Note #8

August 13, 1984

This technical note explains:

1. How to protect auxiliary bank graphics pages from /RAM,
2. How to disconnect and reinstall /RAM (or some other device)

For further information contact:
PCS Developer Technical Support
M/S 22-W. Phone (408) 996-1010

ProDOS TECHNICAL NOTE #8

- Protecting Auxiliary Bank Hi-Res Graphics Pages -
- Disconnecting and Re-installing /RAM -
- Convention on How to Treat Ram Disk's with >64K -

(Revised August 13, 1984)

When ProDOS is booted a check is made of the environment. If a 128K Apple // system is found, the auxiliary 64K bank of memory is configured as a ram disk named /RAM that will appear as slot 3 drive 2 (since it is memory on the 80 column card which appears in slot 3). /RAM's unit number as entered in the ProDOS global page's device list will be $BF.

If you are going to use the auxiliary memory for any other purpose, you must protect yourself from /RAM.

If your use involves hi-res graphics, you may protect those areas of auxiliary memory. If you will save a "dummy" 8K file as the first entry in /RAM it will always be saved at $2000 to $3FFF. If you then immediately save a second "dummy" 8K file to /RAM it will be saved at $4000 to $5FFF. This technique provides a mechanism for protecting the hi-res pages in auxiliary memory while still maintaining /RAM as an online storage device.

There is no formula for determining where the blocks of /RAM physically reside in memory. Further, the logical blocks are not physically contiguous. There is no guaranteed way to protect any other fixed portions of auxiliary memory by the "dummy" file method.

If you wish to protect all of the auxiliary memory that has not been reserved for use by Apple, you must disconnect /RAM. To do this there are three areas of the system global page of interest:

$BF10-$BF2F contains the disk device driver addresses.

$BF31 contains the number of devices minus one.

$BF32-$BF3F contains the list of disk device numbers.

Here are the steps to be followed to disconnect /RAM:

0.) Suggested - Read block two on /RAM and take a peek at the file_count field in the directory. If there are any files on /RAM, prompt the user to continue with the disconnect or abort the process.

1.) Check the MACHID byte at $BF96 to see if you are operating in a 128K environment. If not, there will be no /RAM to disconnect.

2.) The slot 0, drive 1 disk driver vector ($BF10) will point to the "No Device Connected" routine. The slot zero vectors $BF10 and $BF20 ARE RESERVED FOR OUR OWN USE. YOU CANNOT THEREFORE USE THESE VECTORS IF THIS CONVENTION IS TO WORK! If the slot 3 drive 2 vector also points to the same address, then /RAM is already disconnected.

3.) If we have determined that /RAM is on line, we are ready to remove it.

NOTE: If ProDOS has just been booted, /RAM is the last "disk" device installed. However, if the user has "manually" installed another device(s) the device number for /RAM will not be the last entry in the device list (DEVLST).

Also note that the following steps can be generically followed if you wish to disconnect ANY device.

a.) Retrieve the slot 3, drive 2 device number you find in DEVLST and save it.

b.) Move any remaining device numbers forward in the DEVLST.

c.) Retrieve the slot 3 drive 2 driver vector and save it for later re-installation.

d.) Replicate the "No Device Connected" vector in slot 0 drive 1 into slot 3 drive 2.

e.) Decrement the device count (DEVCNT).

/RAM is now disconnected and you are free to use the unreserved areas of auxiliary memory.

A convention has now been established for those ram disks with a capacity greater than 64K and wish not to be disconnected by programs that would not realize excess memory could still be utilized by the ram disk driver.

Here is what the routine might look like:

```
S   E   FILE #01 =>/P/INSTALLRAM
----- NEXT OBJECT FILE NAME IS /P/INSTALLRAM.0
1000:          1000     1             ORG   $1000
1000:          BF31     2 DEVCNT      EQU   $BF31        ; GLOBAL PAGE DEVICE COUNT
1000:          BF32     3 DEVLST      EQU   $BF32        ; GLOBAL PAGE DEVICE LIST
1000:          BF98     4 MACHID      EQU   $BF98        ; GLOBAL PAGE MACHINE ID BYTE
1000:          BF26     5 RAMSLOT     EQU   $BF26        ; SLOT 3, DRIVE 2 IS /RAM'S DRIVER VECTOR
1000:                   6 *
1000:                   7 * NODEV IS THE GLOBAL PAGE SLOT ZERO, DRIVE 1 DISK DRIVE VECTOR.
1000:                   8 * IT IS RESERVED FOR USE AS THE "NO DEVICE CONNECTED" VECTOR.
1000:                   9 *
1000:          BF10    10 NODEV       EQU   $BF10        ;
1000:                  11 *
1000:                  12 * FIRST THING TO DO IS TO SEE IF THERE IS A /RAM TO DISCONNECT!
1000:                  13 *
1000:AD 98 BF          14             LDA   MACHID       ; LOAD THE MACHINE ID BYTE
1003:29 30             15             AND   #$30         ; TO CHECK FOR A 128K SYSTEM
1005:C9 30             16             CMP   #$30         ; IS IT 128K?
1007:D0 4D     1056    17             BNE   DONE         ; IF NOT, THEN BRANCH SINCE NO /RAM!
1009:                  18 *
1009:AD 26 BF          19             LDA   RAMSLOT      ; IT IS 128K; IS A DEVICE THERE?
100C:CD 10 BF          20             CMP   NODEV        ; COMPARE WITH LOW BYTE OF NODEV
100F:D0 08     1019    21             BNE   CONT         ; BRANCH IF NOT EQUAL, DEVICE IS CONNECTED
1011:AD 27 BF          22             LDA   RAMSLOT+1    ; CHECK HI BYTE FOR MATCH
1014:CD 11 BF          23             CMP   NODEV+1      ; ARE WE CONNECTED?
1017:F0 3D     1056    24             BEQ   DONE         ; BRANCH, NO WORK TO DO; DEVICE NOT THERE!
1019:                  25 *
1019:                  26 * AT THIS POINT /RAM (OR SOME OTHER DEVICE) IS CONNECTED IN
1019:                  27 * THE SLOT 3, DRIVE 2 VECTOR.  NOW WE MUST GO THRU THE DEVICE
1019:                  28 * LIST AND FIND THE SLOT 3, DRIVE 2 UNIT NUMBER OF /RAM ($BF).
1019:                  29 * THE ACTUAL UNIT NUMBERS, (THAT IS TO SAY 'DEVICES') THAT WILL
1019:                  30 * BE REMOVED WILL BE $BF, $BB, $B7, $B3.  /RAM'S DEVICE NUMBER
1019:                  31 * IS $BF.  THUS THIS CONVENTION WILL ALLOW OTHER DEVICES THAT
1019:                  32 * DO NOT NECESSARILY RESEMBLE (OR IN FACT, ARE COMPLETELY DIFFERENT
1019:                  33 * FROM) /RAM TO REMAIN INTACT IN THE SYSTEM.
1019:                  34 *
1019:                  35 *
1019:AC 31 BF          36 CONT        LDY   DEVCNT       ; GET THE NUMBER OF DEVICES ONLINE
101C:B9 32 BF          37 LOOP        LDA   DEVLST,Y     ; START LOOKING FOR /RAM OR FACSIMILE
101F:29 F3             38             AND   #$F3         ; LOOKING FOR $BF, $BB, $B7, $B3
1021:C9 B3             39             CMP   #$B3         ; IS DEVICE NUMBER IN ($BF,$BB,$B7,$B3)?
1023:F0 05     102A    40             BEQ   FOUND        ; BRANCH IF FOUND..
1025:88                41             DEY                ; OTHERWISE CHECK OUT THE NEXT UNIT #.
1026:10 F4     101C    42             BPL   LOOP         ; BRANCH UNLESS YOU'VE RUN OUT OF UNITS.
1028:30 2C     1056    43             BMI   DONE         ; SINCE YOU HAVE RUN OUT OF UNITS TO
102A:B9 32 BF          44 FOUND       LDA   DEVLST,Y     ; GET THE ORIGINAL UNIT NUMBER BACK
102D:8D 59 10          45             STA   RAMUNITID    ; AND SAVE IT OFF FOR LATER RESTORATION.
1030:                  46 *
1030:                  47 * NOW WE MUST REMOVE THE UNIT FROM THE DEVICE LIST BY BUBBLING
1030:                  48 * UP THE TRAILING UNITS.
1030:                  49 *
1030:B9 33 BF          50 GETLOOP     LDA   DEVLST+1,Y   ; GET THE NEXT UNIT NUMBER
1033:99 32 BF          51             STA   DEVLST,Y     ; AND MOVE IT UP.
```

```
1036:F0 03   103B              BEQ   EXIT          ; BRANCH WHEN DONE(ZEROS TRAIL THE DEVLST)
1038:C8                        INY                 ; CONTINUE TO THE NEXT UNIT NUMBER...
1039:D0 F5   1030              BNE   GETLOOP       ; BRANCH ALWAYS.
103B:                   *
103B:AD 26 BF           EXIT   LDA   RAMSLOT       ; SAVE SLOT 3, DRIVE 2 DEVICE ADDRESS.
103E:8D 57 10                  STA   ADDRESS       ; SAVE OFF LOW BYTE OF /RAM DRIVER ADDRESS
1041:AD 27 BF                  LDA   RAMSLOT+1     ; SAVE OFF HI BYTE
1044:8D 58 10                  STA   ADDRESS+1     ;
1047:                   *
1047:AD 10 BF                  LDA   NODEV         ; FINALLY COPY THE 'NO DEVICE CONNECTED'
104A:8D 26 BF                  STA   RAMSLOT       ; INTO THE SLOT 3, DRIVE 2 VECTOR AND
104D:AD 11 BF                  LDA   NODEV+1       ;
1050:8D 27 BF                  STA   RAMSLOT+1     ;
1053:CE 31 BF                  DEC   DEVCNT        ; DECREMENT THE DEVICE COUNT.
1056:60                 DONE   RTS                 ; AND RETURN
1057:                   *
1057:00 00              ADDRESS   DW  $0000        ; STORE THE DEVICE DRIVER ADDRESS HERE
1059:00                 RAMUNITID DFB $00          ; STORE THE DEVICE'S UNIT NUMBER HERE
105A:                   *
```

Part of your exit procedure should include code to re-install /RAM so that it is available to the next application. Don't blindly reinstall /RAM...be sure it is off-line first. Applications should not begin by re-installing /RAM since this would preclude passing files from one application to the next in /RAM.

Here is the way to reinstall /RAM (or any general device):

a.) Re-install the device driver address you retrieved and saved as the slot 3 drive 2 vector.

b.) Increment the device count (DEVCNT).

c.) Re-install the device number in the device list (DEVLST).

NOTE: It may be best to re-install the device number as the first entry in the list. If the user has "manually" installed a disk driver, he may assume that since it was the last thing installed that it is still the last one in the list. Therefore, we recommend that you move all the entries in the list down one and re-install the /RAM device number as the first entry.

d.) Finally, set up the parameters for a format request and JSR to the device driver address you have re-installed. The /RAM driver will set up a "virgin" directory and bit map.

Here is what the reinstallation code might look like:

```
105A:              72 *
105A:              73 * THIS IS THE EXAMPLE /RAM INSTALL ROUTINE
105A:              74 *
105A:AC 31 BF      75           LDY   DEVCNT        ; GET THE NUMBER OF DEVICES - 1.
105D:B9 32 BF      76 LOOP1     LDA   DEVLST,Y      ; LOAD THE UNIT NUMBER
1060:29 B0         77           AND   #$B0          ; CHECK FOR SLOT 3, DRIVE 2 UNIT.
1062:C9 B0         78           CMP   #$B0          ; IS IT THE SLOT 3, DRIVE 2 UNIT?
1064:F0 40    10A6 79           BEQ   DONE1         ; IF SO BRANCH.
1066:88            80           DEY                 ; OTHERWISE SEARCH ON...
1067:10 F4    105D 81           BPL   LOOP1         ; LOOP UNTIL DEVLST SEARCH IS COMPLETED
1069:AD 57 10      82           LDA   ADDRESS       ; RESTORE THE DEVICE DRIVER ADDRESS
106C:8D 26 BF      83           STA   RAMSLOT       ; LOW BYTE..
106F:AD 58 10      84           LDA   ADDRESS+1     ; NOW THE
1072:8D 27 BF      85           STA   RAMSLOT+1     ; HI BYTE.
1075:EE 31 BF      86           INC   DEVCNT        ; AFTER INSTALLING DEVICE,INC DEVICE COUNT
1078:AC 31 BF      87           LDY   DEVCNT        ; USE Y FOR LOOP COUNTER..
107B:B9 31 BF      88 LOOP2     LDA   DEVLST-1,Y    ; BUBBLE DOWN THE ENTRIES IN DEVICE LIST
107E:99 32 BF      89           STA   DEVLST,Y      ;
1081:88            90           DEY                 ; NEXT
1082:D0 F7    107B 91           BNE   LOOP2         ; LOOP UNTIL ALL ENTRIES MOVED DOWN.
1084:              92 *
1084:              93 * NOW SET UP A /RAM FORMAT REQUEST
1084:              94 *
1084:A9 03         95           LDA   #3            ; LOAD ACC WITH FORMAT REQUEST NUMBER.
1086:85 42         96           STA   $42           ; STORE REQUEST NUMBER IN PROPER PLACE.
1088:              97 *
1088:AD 59 10      98           LDA   RAMUNITID     ; RESTORE THE DEVICE
108B:8D 32 BF      99           STA   DEVLST        ; UNIT NUMBER IN THE DEVICE LIST
108E:29 F0        100           AND   #$F0          ; STRIP THE DEVICE ID (ZERO LOW NIBBLE)
1090:85 43        101           STA   $43           ; AND STORE THE UNIT NUMBER IN $43.
1092:             102 *
1092:A9 00        103           LDA   #$00          ; LOAD LOW BYTE OF BUFFER POINTER
1094:85 44        104           STA   $44           ; AND STORE IT.
1096:A9 20        105           LDA   #$20          ; LOAD HI BYTE OF BUFFER POINTER
1098:85 45        106           STA   $45           ; AND STORE IT.
109A:             107 *
109A:AD 8B C0     108           LDA   $C08B         ; READ & WRITE ENABLE
109D:AD 8B C0     109           LDA   $C08B         ; THE LANGUAGE CARD WITH BANK 1 ON.
10A0:             110 *
10A0:             111 * NOTE HOW THE DRIVER IS CALLED.  YOU JSR TO AN INDIRECT JMP SO
10A0:             112 * CONTROL IS RETURNED BY THE DRIVER TO THE INSTRUCTION AFTER THE JSR.
10A0:             113 *
10A0:20 A7 10     114           JSR   DRIVER        ; NOW LET DRIVER CARRY OUT CALL.
10A3:AD 82 C0     115           LDA   $C082         ; NOW PUT ROM BACK ON LINE.
10A6:60           116 DONE1     RTS                 ; THAT'S ALL.
10A7:             117 *
10A7:6C 26 BF     118 DRIVER    JMP   (RAMSLOT)     ; CALL THE /RAM DRIVER
```

The above routines address the specific case of /RAM. However, with a little massaging, they can easily be adapted to install or remove any disk driver routines.

The routines described in this document are examples only. No guarantee is made regarding their performance or suitability for any particular use.

ProDOS TECHNICAL NOTE #9

Buffer Management using BASIC.SYSTEM

(31 August 1983)

BASIC.SYSTEM provides buffer management for file I/O. Those facilities can also be utilized from machine language modules operating in the ProDOS/AppleSoft environment to provide protected areas for code, data, etc.

BASIC.SYSTEM resides from $9A00 upward with a general purpose buffer from $9600 (himem) to $99FF. When a file is opened, BASIC.SYSTEM does garbage collection, if needed, moves the general purpose buffer down to $9200 and installs a file I/O buffer at $9600. When a second file is opened, the general purpose buffer is moved down to $8E00 and a second file I/O buffer is installed at $9200. If an EXEC file is opened, it is always installed as the highest file I/O buffer at $9600, and all the other buffers are moved down. Additional regular file I/O buffers are installed by moving the general purpose buffer down and installing it below the lowest file I/O buffer. All file I/O buffers, including the general purpose buffer, are 1K (1024 bytes) and begin on a page boundary.

BASIC.SYSTEM may be called from machine language to allocate any number of pages (256 bytes) as a buffer, located above himem and protected from AppleSoft Basic programs. The ProDOS bit-map is not altered so that files may be BLOADed into the area without an error from the ProDOS kernel. If you subsequently alter the bit-map to protect the area, it is your responsibility to mark the area as free when you are finished...BASIC.SYSTEM will not do it for you.

To allocate a buffer, simply place the number of desired pages in the accumulator and JSR GETBUFR ($BEF5). If the carry flag returns clear, the allocation was successful and the accumulator will return the high byte of the buffer address. If the carry flag returns set, an error has occurred and the accumulator will return the error code. Note that the X and Y registers are not preserved.

The first buffer is installed as the highest buffer, just below BASIC.SYSTEM, from $99FF downward, regardless of the number and type of file I/O buffers that are open. If a second allocation is requested, it will be installed immediately below the first. Thus, it is possible to assemble code to run at known addresses...relocatable modules are not needed.

To deallocate the buffers created by the above call, it is only necessary to JSR FREEBUFR ($BEF8) and all of the buffers will be deallocated and the file buffers will be moved back up. It is important to note that although more than one buffer may be allocated by this call, they may not be selectively deallocated.

APPLE COMPUTER, Inc. PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #10

Installing Clock Driver Routines in ProDOS

(Revised 8 November 1983)

In you wish to support clock cards other than the ThunderClock, there are a number of possible places to locate your code. The "cleanest" place is to replace the ThunderClock routines located in ProDOS with your routines, if your code will fit.

When the PRODOS system file is executed, it installs the address of the ThunderClock routines at $BF07,$BF08 whether a card is present or not. The address is preceeded with a $4C (JMP) if a ThunderClock card is found or a $60 (RTS) if it was not.

The ThunderClock card is identified by looking at the $Cn00 ROM for:

$Cn00 = $08 $Cn02 = $28 $Cn04 = $58 $Cn06 = $70

If you look at $BF07,$BF08 you will find the location to put your code. There is room for 125 bytes.

To install your code, simply write enable the "language card" area, and move your code. Don't forget that your relocation code must justify the absolute addresses as part of the relocation procedure. Finally, restore any soft-switches you have changed. (There is no guarantee as to the absolute location of the clock driver code on future revisions of ProDOS, only that it's location may be found by examining the global page, as mentioned above.)

All that your code need do is get the time from the clock card, convert it to the ProDOS format and store it in the date and time locations in the global page.

Your installation routine can be called from an application or as part of the STARTUP program.

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #11

The ProDOS Machine Identification Byte

* THIS NOTE SUPERCEDES THE INFORMATION *
* FOUND IN SECTIONS 5.2.3 & 5.3.1 OF THE *
* ProDOS TECHNICAL REFERENCE MANUAL *

(revised 08 May 1984)

The Machine Identification byte (MACHID) in the ProDOS system global page has been redefined to permit identification of future products from Apple Computer, Inc. that may use the ProDOS operating system. The change does not impact any checking for existing systems that your application may now be doing.

The definition of MACHID at $BF98 is:

Bits 7-6	If bit 3 = 0 then
	00 =][
	01 =][+
	10 = //e
	11 = /// emulation
		If bit 3 = 1 then
		00 = reserved
		01 = reserved
		10 = //c
		11 = reserved

Bits 5-4	00 = reserved, 01 = 48K, 10 = 64K, 11 = 128K	
Bit 3	The value of bit 3 determines how bits 7-6 will be interpreted. See Bits 7-6 definition.	
Bit 2	Reserved for future definition	
Bit 1	0 = No 80-column card 1 = 80-column card installed	
Bit 0	0 = No ThunderClock or equivalent 1 = ThunderClock or equivalent installed	

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 996-1010

ProDOS TECHNICAL NOTE #12

Interrupt Handling

(1 December 1983)

This technical note expands upon the information found in the ProDOS Technical Reference Manual. It is assumed that the reader has already read and understands the sections regarding interrupts.

This tech note includes a superior example of an interrupt handler for use with ProDOS. The example in the book works properly, however, it will always claim every interrupt whether it came from the clock or not. Additionally, it does not conform to one protocol which will be required in future revisions of ProDOS, nor does it incorporate some common examples of good programming practice.

Vectors for interrupt handlers must be installed and removed with ALLOC_INTERRUPT and DEALLOC_INTERRUPT calls to ProDOS. Even though the vectors appear in the system global page, you must always use only the systems calls...never change the global page entries yourself.

All interrupt routines must commence with a CLD instruction. Although not checked in the initial release of ProDOS, this first byte will be checked in future revisions to verify the validity of the interrupt handler.

Good programming practice dictates that an interrupt handler should preserve the status register (PHP) and mask interrupts (SEI). The code should restore the status register (PLP) before exit, and before setting or clearing the carry flag as required by ProDOS.

If your application includes an interrupt handler, before you exit:

(1) Turn off the interrupts...remember, an unclaimed interrupt will cause system death.

(2) Make a DEALLOC_INTERRUPT call before exiting from your application. Don't leave a vector installed that will point to a routine that is gone.

Within your interrupt handler routines, you MUST leave ALL memory banks in the same configuration you found them. DON'T FORGET ANYTHING...main language card, main alternate $D000, main motherboard ROM...and, on an Apple //e...auxiliary language card, auxiliary alternate $D000, alternate zero page and stack, etc., etc... This is important! The ProDOS interrupt receiver assumes the environment is absolutely unaltered when your handler relinguishes control.

If your handler recognizes the interrupt and services it, the carry should be cleared (CLC) immediately before returning (RTS). If it was not your interrupt, the carry should be set (SFC) immediately before returning (RTS). Do not use a return from interrupt (RTI) to exit...the ProDOS interrupt receiver still has some housekeeping to perform before it issues the RTI instruction.

Here is a sample routine which will turn on interrupts on a ThunderClock card and print the date and time to the upper right corner of the screen.

```
0300:          0300          ORG $300
0300:          C20B  WTTCP   EQU $C20B     ; Clock write entry point (Slot 2)
0300:          C208  RDTCP   EQU $C208     ; Clock read entry point (Slot 2)
0300:          C080  TCICR   EQU $C080     ; Interrupt cont. register (Slot 2)
0300:          C088  TCMR    EQU $C088     ; Mystery register (Slot 2)
0300:                *
0300:          0200  IN      EQU $200      ; Where the clock leaves the time
0300:                *
0300:          0412  UPRIGHT EQU $412      ; The upper right of the screen
0300:          047A  INTON1  EQU $47A      ; Leave interrupts on (Slot 2)
0300:          07FA  INTON2  EQU $7FA      ; Leave interrupts on (Slot 2)
0300:                *
0300:          BF00  MLI     EQU $BF00     ; Entry point to the ProDOS MLI
0300:                *
0300:                * CALLING INTERRUPTS, CALLING INTERRUPTS
0300:                *
0300:20 7E 03                JSR ALLOC.INT ; Install interrupt routine
0303:60                      RTS           ; That's all forks
0304:                *
0304:                *
0304:          0304  SHOWTIME EQU *
0304:D8                      CLD
0305:08                      PHP
0306:78                      SEI           ; Disable Interrupts
0307:A0 20                   LDY #$20      ; For slot 2
0309:B9 80 C0                LDA TCICR,Y   ; Get Interrupt Control Reg value
030C:29 20                   AND #$20      ; Bit 5 indicates INT is clock
030E:F0 3C     034C          BEQ NOTCLK    ; If bit 5 is off, not from clock
0310:B9 88 C0                LDA TCMR,Y    ; Clear mystery register
0313:B9 80 C0                LDA TCICR,Y   ; Clear interrupt on hardware
0316:CE 4F 03                DEC COUNTER   ; Only print time every second
0319:D0 2E     0349          BNE EXITCLK   ; Not time to print yet
031B:                *
031B:A2 27                   LDX #39       ; Save the input buffer
031D:BD 00 02        DOIN    LDA IN,X      ; Since the clock writes over it
0320:9D 56 03                STA INBUF,X   ; When it is called
0323:CA                      DEX
0324:10 F7     031D          BPL DOIN
0326:                *
0326:A9 A5                   LDA #$A5      ; Set Applesoft string input mode
0328:20 0B C2                JSR WTTCP     ; and send it to the card
032B:20 08 C2                JSR RDTCP     ; Read time into input buffer
032E:                *
032E:A2 15                   LDX #21
0330:BD 01 02        GETNEXT LDA IN+1,X    ; Print time to screen
0333:9D 12 04                STA UPRIGHT,X ; Chars 0-22 of input buffer
0336:CA                      DEX
0337:10 F7     0330          BPL GETNEXT
0339:                *
0339:A9 40           SETCNTR LDA #64       ; Set up counter for next time
033B:8D 4F 03                STA COUNTER
033E:                *
033E:A2 27                   LDX #39       ; Restore the input buffer
0340:BD 56 03        DOIN2   LDA INBUF,X
0343:9D 00 02                STA IN,X
0346:CA                      DEX
0347:10 F7     0340          BPL DOIN2
```

```
0349:                 *
0349:28               EXITCLK    PLP          ; Tell MLI we processed the INT
034A:18                          CLC
034B:60                          RTS
034C:28               NOTCLK     PLP
034D:38                          SEC          ; Tell MLI it isn't ours
034E:60                          RTS
034F:                 *
034F:         0001    COUNTER    DS  1,0
0350:                 *
0350:02 00            AIPARMS    DFB 2,0      ; Put allocate and deallocate
0352:04 03                       DW  SHOWTIME ; Interrupt parameters here
0354:                 *
0354:01 00            DIPARMS    DFB 1,0      ; so both routines can use them
0356:                 *
0356:         0028    INBUF      DS  40,0     ; Save 40 bytes of IN here
037E:                 *                       ; for input buffer save/restore
037E:                 *
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
037E:20 00 BF         ALLOC.INT JSR MLI       ; Call the MLI
0381:40                          DFB $40      ; to allocate the interrupt
0382:50 03                       DW  AIPARMS
0384:D0 19    039F               BNE OOPS     ; Break on error
0386:                 *
0386:A0 20                       LDY #$20
0388:A9 AC                       LDA #$AC     ; Set 64hz interrupt rate
038A:20 0B C2                    JSR WTTCP    ; by writing a ',' to clock
038D:A9 40                       LDA #$40     ; Now enable the software
038F:8D 7A 04                    STA INTON1   ; and tell it not to disable
0392:8D FA 07                    STA INTON2   ; interrupts after reads
0395:99 80 C0                    STA TCICR,Y
0398:A9 01                       LDA #1       ; Print time immediately
039A:8D 4F 03                    STA COUNTER  ; Once per second later
039D:58                          CLI          ; Allow the 6502 to see the
039E:60                          RTS          ; interrupts
039F:                 *
039F:00               OOPS       BRK          ; Break on error
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
03A0:A9 00            DEALLOC.INT LDA #0      ; Disable interrupts
03A2:8D 7A 04                    STA INTON1   ; in the thunder clock
03A5:8D FA 07                    STA INTON2
03A8:A0 20                       LDY #$20
03AA:99 80 C0                    STA TCICR,Y
03AD:                 *
03AD:AD 51 03                    LDA AIPARMS+1 ; GET INT_NUM
03B0:8D 55 03                    STA DIPARMS+1 ; FOR DEALLOCATION
03B3:20 00 BF                    JSR MLI       ; CALL THE MLI
03B6:41                          DFB $41       ; TO DEALLOCATE THE INTERRUPT
03B7:54 03                       DW  DIPARMS   ; POINTER TO PARAMETER LIST
03B9:D0 01    03BC               BNE OOPS2     ; BREAK ON ERROR
03BB:60                          RTS           ; DONE
03BC:                 *
03BC:00               OOPS2      BRK           ; BREAK ON ERROR
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

ProDOS TECHNICAL NOTE #13

Double High Resolution Graphics Files

(6 January 1984)

The 128K Apple //e supports a graphics mode known as Double Hi-Res Graphics in which both main and auxiliary memory hi-res graphics pages are used to produce pictures with twice as many dot positions horizontally.

Apple /// graphics has a similar mode and a FOTOFILE file type ($08) has been defined under SOS to contain the screen image. All 16K double hi-res files under ProDOS should be of this file type.

The format of the file is as shown at the right. The "graphics mode" is stored in the 121st byte of the file (Location $78 in the file). The modes for both 1st and 2nd page of double hi-res are:

			Pg 1	Pg 2
280 X 192	Limited Color	=	1	5
560 X 192	Black and White	=	2	6
140 X 192	Full Color	=	3	7

```
                  End-of-file
$3FFF|-------------------|
     |    Main Memory    |
     |  portion of file  |
$2000|___________________|
$1FFF|                   |
     | Auxiliary  Memory |
     |  portion of file  |
$0000|-------------------|
       Beginning of file
```

The normal Apple][hi-res 280 X 192 screen may be BSAVEd as usual. If you desire, for Apple /// SOS compatibility, you may also save these screens as an 8K type $08 FOTOFILE and mark the graphics mode as zero (page 1) or four (page 2), (Apple /// 280 X 192 Black and White mode).

APPLE COMPUTER, Inc., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone (408) 554-5213 or (408) 996-1010

ProDOS TECHNICAL NOTE #14

Selector/Dispatcher Conventions

(revised 09 March 1984)

ProDOS MLI call $65, the QUIT call moves addresses $D100 - $D3FF from the second 4K-byte bank of RAM of the language card to $1000 and executes a JMP to $1000. What initially resides in that area is OUR dispatcher code.

The dispatcher once executed does the following:

1. Interactively allows you to enter a prefix and file name of the system program (interpreter) that you wish to execute.

2. Stores the system program name at $280 starting with a length byte. This is done so once the system program executes, it can find from where is was started and locate any files it could need for processing.

3. Closes any open files.

4. Clears the bit map and protects the zero, stack, text and ProDOS Global pages.

5. Reads in the system file at $2000 and executes a JMP to $2000.

If you wish, you can install your own QUIT code which may load in your own full blown selector program. If you choose to do this, you must at some point:

1. Follow steps 2 - 4 above.

2. THE $D100 BYTE MUST BE A CLD ($D8) INSTRUCTION. This convention is established so programs will be able to tell whether it is selector code or the ProDOS dispatcher code that is resident.

In addition to just leaving the pathname at $280 for the interpreters own use, a method to enable a selector program to specify an accomanying 'STARTUP' program has been defined. Once active, an interpreter can immediately run that program.

The procedure will be to reserve an area in the system file which will be overwritten by a selector program with the 'STARTUP' programs name. The interpreter would then load and execute that specified program.

The actual nuts and bolts of this procedure are as follows:

The selector program will look at the first byte of the interpreter at $2000. If it is a JMP ($4C) instruction, and bytes $2003 and $2004 are both $EE's, then byte $2005 will be interpreted as a buffer size indicator with the buffer starting at $2006. The string at $2006 would be the normal ProDOS pathname or partial pathname starting with a length byte.

JMP CONT	$2000-$2002
$EE | $EE	$2003-$2004
$41 (eg.)	$2005
$07	$2006
STARTUP	$2007-$200D
:	
CONT: (eg.)	$2047

The two $EE's serve as a marker to the selector program to let it know that this particular interpreter can run a startup program. The interpreters that will support this feature will of course supply their own default string which may be a startup program or a flag of your own choice.

For more information on Interpreter Conventions please see ProDOS Technical Note #7.

APPLE COMPUTER INC., PCS Developer Technical Support
20525 Mariani Avenue, M/S 22-W
Cupertino, CA 95014
Phone: (408) 996-1010