DISK EJECTOR

Just like the Finder, your Applesoft programs can eject 3.5-inch disks. This short machine-language utility works under DOS 3.3 and ProDOS.

Rescue your 3.5-inch disks from the crocodile's maw. With just a simple CALL statement, your Applesoft programs can now eject the disk out of a UniDisk 3.5 or an Apple 3.5 drive in both the ProDOS and DOS 3.3 environments. When your program encounters a write-protected disk, it can eject the offending disk instead of just beeping at you.

You can customize your program's QUIT option with Disk Ejector to eject all 3.5-inch disks before exiting. No more accidentally leaving the disks in the drives when you turn your computer off for the evening. All it takes is one CALL to the eject routine for each 3.5-inch drive attached.

EJECTING A DISK

Two values need to be passed to the eject routine in order for it to eject your 3.5-inch disk. These values are the slot number the drive is attached to and the number of the drive. The syntax for this CALL statement is:

CALL EJ,S,D

where EJ is the address at which the eject routine starts in memory, S is the slot number, and D is the drive number. When this statement is executed, the 3.5-inch drive's light comes on, the ejection mechanism whines, and the disk pops out.

As an example, let's assume that you want your program to eject the disk in slot 5, drive 1, and you have loaded the routine at location 32768. Your CALL would look like this:

CALL 32768,5,1

If you want to eject the disk in drives 1 and 2, your program segment would be as follows:

```
1000 S = 5
1010 D = 1
1020 EJ = 32768
1030 CALL EJ,S,D
1040 D = 2
1050 CALL EJ,S,D
```

That's all it takes!

ENTERING THE PROGRAM

To enter the disk ejector routine, key in the assembly language source code in **Listing 1** and assemble it. If you do not have an assembler, enter the Monitor with a CALL −151, key in the hex code found in **Listing 2**, and save it using the command:

BSAVE DISK.EJECTOR,A$8000,L$115

Then key in the demonstration program found in **Listing 3** and save it with the command:

SAVE EJECT.DEMO

For help with entering *Nibble* listings, see the Typing Tips section.

HOW THE PROGRAM WORKS

The original method of ejecting 3.5-inch disks was presented by Tom Weishaar in his *Open-Apple* newsletter (Vol. 2/No. 5, page 2.38). His version was put into memory by way of the Lam method, could not easily be moved to any other area of memory, and required you to POKE in the slot and drive numbers. The program presented here can be BLOADed into memory, is fully relocatable, and allows the user to pass the slot and drive numbers as variables. It also prevents the user from accidentally requesting a slot number higher than seven or reformatting a SCSI hard disk drive.

This program must modify several portions of itself, including the parameter table, in order to be relocatable. It first stores the address where it started (the address of the last CALL can be found at locations $50 and $51). The program then deciphers what the desired slot number is, verifies that it's less than eight, gets the entry point for the desired slot, and then modifies the address of the JSR command (**Listing 1, line 130**). The entry point is found by first getting the value at $C*s*FF (where *s* is the slot number) and adding it to $C*s*03.

Next, it determines the desired drive number and stores it. Before the program goes any further, it checks the requested slot to verify that a Smartport card (such as the IIc's or IIGS's disk port, or a UniDisk 3.5-inch controller card) is present. If any other card (Disk II, SCSI, and so on) is detected, a special error code is stored in location 255 and control is returned to the calling program. This traps most, but not all, possible misuses of this routine. Another check is done to verify that the requested drive is indeed a 3.5-inch drive, which is necessary because devices other than a 3.5-inch drive can be connected to the IIc's or the IIGS's disk port. If a non-3.5-inch drive is requested, then a special error code is stored in location 255 and control is returned to the calling program.

Finally, the parameters are passed to the drive and the disk is ejected. If an error (such as "DEVICE NOT CONNECTED") occurs, then its error number is saved in location 255 (an unused zero page location) before control is returned to the user.

Since this routine actually bypasses the disk operating system and communicates directly with the disk controller card, it works under both ProDOS and DOS 3.3. However, Apple IIGS owners should be careful when using this routine. ProDOS 8 will remap the third

and fourth 3.5-inch drives attached to the Smartport to slot 2. The disk ejector works with physical addresses, not ProDOS' logical addresses. This means that the disk ProDOS, considers to be in slot 2, drive 1 is considered to be in slot 5, drive 3 by the disk ejector.

If you have the IIGS RAM disk enabled and have two 3.5-inch drives connected to the Smartport, keep in mind that the RAM disk is slot 5, drive 2. This means that the second drive is actually slot 5, drive 3 to the disk ejector.

If you try to eject a disk from a drive that doesn't exist, a "DEVICE NOT CONNECTED" error (number 40) will be returned to your program. Try ejecting the disk in slot 5, drive 20 to verify this error. You may check for errors by PEEKing memory location 255 ($00FF) after the CALL. Always remember to store a zero here either after checking or before the CALL, to ensure that errors are cleared between occurrences.

If the slot number passed to the ejector is greater than seven, then a "SYNTAX ERROR" message will be returned and location 255 will contain a special error code (203). The same error message is also generated if you leave out a comma or a variable in the CALL statement. Since location 255 will contain an unknown value, always check both the regular error number (location 222) and the special error number (location 255).

MODIFICATIONS

This routine is self-modifying, so any changes made must be handled carefully. Make sure that the upper portion of this routine is still accessible through the use of zero page indexed addressing. If you add any additional features, you may need to either modify the address mode used for the self-modification or place your additions at the end of the routine.

Possible modifications include setting slot 5 as the default, or making this routine work as an ampersand routine for use with the Beagle Bros ProDOS Compiler or Roger Wagner's ampersand utilites. To use it with Roger Wagner's toolbox series of ampersand commands, you must eliminate the JSR CHKCOM from **line 93** and change the locations from which the starting address is read. If you are using an assembler to make these changes, simply delete **line 93** and change the locations in **lines 62 and 64** from $50 and $51 to $5E and $5F. If you are making these changes without an assembler, then follow these steps:

```
BLOAD DISK.EJECTOR,A$8000
POKE 32769,94
POKE 32773,95
POKE 32789,234
POKE 32790,234
POKE 32791,234
BSAVE DISK.EJECTOR,A$8000,L$115
```

Be sure to make these changes on a backup copy of this routine — just in case something goes wrong, goes wrong, goes wrong....

LISTING 1: DISK.EJECTOR.S

```
1  ****************************
2  *                          *
3  *   DISK EJECTOR FOR APPLE   *
4  *    3.5 INCH DISK DRIVES.   *
5  *                          *
6  *    Syntax:  CALL EJ,S,D    *
7  *                          *
8  *    EJ= ADDRESS OF ROUTINE  *
9  *    S = SLOT # (1-7)        *
10 *    D = DRIVE # (1-127)     *
11 *    BOTH MUST BE GIVEN!!!!! *
12 *                          *
13 *       BY Tim Swihart       *
14 *                          *
15 *     COPYRIGHT (C) 1988     *
16 *     by MicroSPARC, INC.    *
17 *     Concord, MA  01742     *
18 *                          *
19 *      MERLIN PRO V2.54      *
20 *                          *
21 ****************************
22 *
23          ORG   $8000
24 *
25 CsFF     EQU   $C0FF       ;Address of requested slot
26 *                           (rewritten during execution)
27 CHKCOM   EQU   $DEBE       ;Checks for comma
28 FRMNUM   EQU   $DD67       ;Evaluates passed formula
29 GETADR   EQU   $E752       ;Converts # in FAC to an integer
30 LINNUM   EQU   $0050       ;Lo-byte of integer from GETADR
31 DRIVE    EQU   $00FA       ;Storage for requested drive #
32 SLOT     EQU   $00FB       ;Stores address for slot #
33 US       EQU   $00FD       ;Stores address for start of
34 *                           this routine
35 REWRIT   EQU   $0000       ;This value will be rewritten
36 *                           during execution
37 USERERR  EQU   $00FF       ;Store error code here when done
38 CMD      EQU   $00         ;0 = Status, 4 = Control
39 DRV      EQU   $00         ;Smartport drive #
40 SUBCMD   EQU   $00         ;3 = Request DIB
41 SIZE     EQU   $00         ;Number of bytes in CTL
42 TYPE     EQU   $00         ;Type = $01 for 3.5" disk
43 *
44 * SPECIAL ERROR CODES
45 BADDRIV  EQU   $CA         ;Requested drive NOT a 3.5"
46 BADSLOT  EQU   $CB         ;Requested slot higher than 7
47 BADCARD  EQU   $CC         ;Requested slot not a Smartport
48 *
49 * Any line beginning with the label 'Pxx' (where xx is
50 * a number) MUST be present when you assemble this program.
51 * These labels are used to determine offsets that are
52 * REQUIRED for this program to work properly.
53 *
54 *
55 * First, we buffer off the address where this routine
56 * is currently located and store it at 'US'.  If you want
57 * to modify this routine to work from a language other
58 * than APPLESOFT, then you will have to replace the next
59 * four lines of code with a different method for deter-
60 * mining where this routine starts in memory.
61 *
62 START    LDA   $50         ;Lo-byte of where routine starts
63          STA   US          ;Buffer it to $00FE
64          LDA   $51         ;Hi-byte of where routine starts
65          STA   US+1        ;Buffer it to $00FF
66 *
67 *
68 * The starting address is needed so that this routine can
69 * use zero-page indexed addressing to access portions of
70 * itself.  Since the last portion of this program is where
71 * the "self-referencing" is taking place, and since this
72 * routine is more than 255 bytes long (the upper limit for
73 * zero-paged indexed addressing is 255), we must adjust
74 * 'US'.  This makes sure that all of the upper portion of
75 * this routine can be accessed by means of zero-page
76 * indexed adressing.
77 *
78          CLC
79          LDA   US          ;Get lo-byte of 'US'
80          ADC   #PAS-START  ;Add adjustment
81          STA   US          ;Save new lo-byte of 'US'
82          LDA   US+1        ;Get hi-byte of 'US'
83          ADC   #$00        ;Adjust hi-byte
84          STA   US+1        ;Save new hi-byte of 'US'
85 *
86 *
87 * Now eat the comma that lives between 'EJ' and 'S' in the
88 * user's 'CALL EJ,S,D' command.  If you want to use this
89 * routine with an ampersand handler that eats the comma for
90 * you, then either remove the following line or replace it
91 * with three NOP's.
92 *
93          JSR   CHKCOM      ;'SYNTAX ERROR' if no comma
94 *
95 *
96 * Evaluate the slot passed from the calling program.  Make
97 * sure that it is less than 8 (since the highest numbered
98 * slot is slot 7).  Returns with a 'SYNTAX ERROR' and puts
99 * a special error code into location $00FF if the slot
100 * number is higher than seven.  If there is an syntax error
101 * in the expresion for d location $00FF contains a
100 * number is higher than seven.  If there is an syntax error
101 * in the expresion for 'S', then a 'SYNTAX ERROR' is
102 * returned and location $00FF contains garbage left by
103 * another routine.
104 *
105          JSR   FRMNUM      ;Evaluate 'S'
106          JSR   GETADR      ;Convert it to an integer and
107 *                           store at locations $50 and $51
108          LDA   LINNUM      ;Get lo-byte from $50
109          CMP   #$08        ;Is slot between 0 and 7?
110          BCC   CONT        ;Yes, so continue
111          LDA   #BADSLOT    ;No, so prepare special error
112          STA   USERERR     ;Save it for the user
```

LISTING 1: DISK.EJECTOR.S *(continued)*

```
113           RTS               ;Return to calling program
114 CONT      ADC   #$C0        ;Prepare '$Cs' where 's' is slot
115           STA   SLOT+1      ;Store as hi-byte of 'SLOT'
116           LDA   #$00        ;Prepare lo-byte of 'SLOT'
117           STA   SLOT        ;'SLOT' now contains $Cs00
118 *
119 *
120 * Now eat the comma between 'S' and 'D' of the CALL from
121 * the user.
122 *
123           JSR   CHKCOM      ;'SYNTAX ERROR' if no comma
124 *
125 *
126 * Evaluate the drive number that the user passed to us.
127 * If there is a syntax error in the expression for 'D',
128 * then a 'SYNTAX ERROR' is returned and location $00FF
129 * contains garbage left by some other routine.
130 *
131           JSR   FRMNUM      ;Evaluate 'D'
132           JSR   GETADR      ;Convert it to an integer and
133 *                            store at locations $50 and $51
134           LDA   LINNUM      ;Get the lo-byte
135           STA   DRIVE       ;Save it for later
136 *
137 *
138 * Verify that the requested slot contains a Smartport.
139 * If not then return the special error code for 'BADCARD'.
140 *
141           LDY   #$01        ;First signature byte is at $Cs01
142           LDA   (SLOT),Y    ;Get signature byte #1
143           CMP   #$20        ;Is it correct?
144           BEQ   NEXT1       ;Yes, go check signature byte #2
145           LDA   #BADCARD    ;No, so prepare special error
146           STA   USERERR     ;Save it for the user
147           RTS               ;Return to calling program
148 NEXT1     LDY   #$03        ;Next signature byte is at $Cs03
149           LDA   (SLOT),Y    ;Get signature byte #2
150           CMP   #$00        ;Is it correct?
151           BEQ   NEXT2       ;Yes, go check signature byte #3
152           LDA   #BADCARD    ;No, so prepare special error
153           STA   USERERR     ;Save it for the user
154           RTS               ;Return to calling program
155 NEXT2     LDY   #$05        ;Next signature byte is at $Cs05
156           LDA   (SLOT),Y    ;Get signature byte #3
157           CMP   #$03        ;Is it correct?
158           BEQ   NEXT3       ;Yes, go check signature byte #4
159           LDA   #BADCARD    ;No, so prepare special error
160           STA   USERERR     ;Save it for the user
161           RTS               ;Return to calling program
162 NEXT3     LDY   #$07        ;Next signature byte is at $Cs07
163           LDA   (SLOT),Y    ;Get signature byte #4
164           CMP   #$00        ;Is it correct?
165           BEQ   PAS         ;Yes, so continue
166           LDA   #BADCARD    ;No, so prepare special error
167           STA   USERERR     ;Save it for the user
168           RTS               ;Return to calling program
169 *
170 *
171 * Now that we know we're talking to a Smartport, we need
172 * to set up the command table to talk to the drive so
173 * that we can verify it is a 3.5" drive.  To do this, we
174 * need to calculate the Smartport entry point.  This is
175 * always three more than the Prodos entry point.  The
176 * Prodos entry point is found by adding the value in $CsFF
177 * to $Cs00.
178 *
179 PAS       LDY   #$FF        ;Prepare to get byte at $CsFF
180           LDA   (SLOT),Y    ;Get value at $CsFF
181           CLC
182           ADC   #$03        ;Add three to it
183           STA   SLOT        ;Save it in lo-byte of 'SLOT'
184 *
185 * Pay attention, things get tricky here.  First, we modify
186 * the 'REWRIT' portion of 'JSR REWRIT'.  (The accumulator
187 * already holds the lo-byte of the Smartport's entry
188 * point).
189 *
190           LDY   #DOIT-PAS+1 ;Offset to middle byte of
191 *                            'JSR REWRIT'
192           STA   (US),Y      ;Put lo-byte of Smartport entry
193 *                            point into 'JSR REWRIT'
194           LDA   SLOT+1      ;Get hi-byte of Smartport entry
195           INY               ;Adjust offset
196           STA   (US),Y      ;Put hi-byte of Smartport entry
197 *                            point into 'JSR REWRIT'
198 *
199 * Now, modify the address of the command list (CMDLIST).
200 *
201           LDA   US          ;Get lo-byte of this routine's
202 *                            starting address
203           LDY   #P3-PAS     ;Offset to pointer to CMDLIST
204           CLC
205           ADC   #CMDLIST-PAS ;Offset to CMDLIST itself
206           STA   (US),Y      ;CMDLIST pointer's lo-byte
207           LDA   US+1        ;Get ready to do hi-byte
208           ADC   #$00        ;Hi-byte of CMDLIST's pointer
209           INY               ;Adjust the offset
210           STA   (US),Y      ;CMDLIST pointer's hi-byte
211 *
212 * Now, we modify the address of the control list (CTL).
213 *
214           LDA   US          ;Get lo-byte of this routine's
215 *                            starting address
216           LDY   #P5-PAS     ;Offset to pointer to CTL
217           CLC
218           ADC   #CTL-PAS    ;Offset to CTL itself
219           STA   (US),Y      ;Rewrite CTL pointer's lo-byte
220           LDA   US+1        ;Get ready to do hi-byte
221           ADC   #$00        ;Hi-byte for CTL's pointer
222           INY               ;Adjust offset
223           STA   (US),Y      ;Rewrite CTL pointer's hi-byte
224 *
225 * Now, we modify the address portion of 'JSR DO_IT'.
226 *
227           LDY   #P1-PAS+1   ;Offset to middle byte of
228 *                            'JSR DO_IT'
229           LDA   US          ;Get lo-byte of this routine's
230 *                            starting address
231           CLC
232           ADC   #DOIT-PAS   ;Offset to 'JSR REWRIT'
233           STA   (US),Y      ;Rewrite lo-byte of address
234 *                            portion of 'JSR DO_IT'
235           LDA   US+1
236           ADC   #$00        ;Prepare hi-byte
237           INY               ;Adjust offset for hi-byte
238           STA   (US),Y      ;Rewrite hi-byte of address
239 *                            portion of 'JSR DO_IT'
240 *
241 *
242 * Things get a little easier now as we set up the drive #,
243 * command #, subcommand # and the size of the control list
244 * so that we can request the Device Information Block (DIB)
245 * from the drive the user sent.
246 *
247           LDA   DRIVE       ;Get the drive number
248           LDY   #P4-PAS     ;Offset to DRV in CMDLIST
249           STA   (US),Y      ;Rewrite the drive number
250 *
251           LDA   #$00        ;CMD = 0 for STATUS call
252           LDY   #P2-PAS     ;Offset to CMD
253           STA   (US),Y      ;Rewrite CMD
254 *
255           LDA   #$03        ;SUBCMD = 3 to request DIB
256           LDY   #P6-PAS     ;Offset to SUBCMD
257           STA   (US),Y      ;Rewrite SUBCMD
258 *
259 *
260 * Now get the DIB from the drive and make sure the drive
261 * is a 3.5" (Unidisk 3.5 or Apple 3.5).  If the drive is
262 * NOT a 3.5" drive, then a special error code is placed
263 * in location $00FF and control is returned to the
264 * calling program.  If the device does not exist, or some
265 * other error occurs, then that error is placed in
266 * location $00FF and control is returned to the calling
267 * program.
268 *
269 P1        JSR   DOIT        ;Gets DIB for us
270           BCC   WORKED      ;If no error then continue
271           STA   USERERR     ;Else save error
272           RTS               ;And go back to calling program
273 WORKED    LDY   #P7-PAS     ;Offset to device type in CTL
274           LDA   (US),Y      ;Gets TYPE from CTL
275           CMP   #$01        ;Is it a 3.5" drive?
276           BEQ   MORE        ;Yes, so keep going
277           LDA   #BADDRIV    ;No, so prepare special error
278           STA   USERERR     ;Save it for the user
279           RTS               ;Return to calling program
280 *
281 *
282 * Now, we have to change the command #, subcommand #,
283 * the size of the control list and make sure the first
284 * byte in the control list is zero.
285 *
286 MORE      LDA   #$04        ;CMD=4 for CONTROL call
287           LDY   #P2-PAS     ;Offset to CMD
288           STA   (US),Y      ;Rewrite CMD
289 *
290 * SUBCMD=4 for eject and accumulator already contains 4
291 *
292           LDY   #P6-PAS     ;Offset to SUBCMD
293           STA   (US),Y      ;Rewrite SUBCMD
294 *
295           LDA   #$00        ;Control list contains zero
296 *                            bytes for eject command
297           LDY   #CTL-PAS    ;Offset to SIZE in CTL
298           STA   (US),Y      ;Rewrite SIZE
299           INY               ;Adjust offset
300           STA   (US),Y      ;Make first byte in CTL equal 0
301 *
302 *
303 * Believe it or not, we're finally ready to eject the
304 * disk!!!  Whew, what a ride!!
305 *
306 DOIT      JSR   REWRIT      ;This line kicks the disk out!
307 P2        DB    CMD         ;0 for STATUS, 4 for CONTROL
308 P3        DA    CMDLIST     ;Address of the command list
309 *                            will be rewritten here
310           STA   USERERR     ;Save any error for the user
311 *                            A zero will be here if no error
312           RTS               ;FINISHED!!!!!!!!
313 *
314 *
315 * Here are the Command List (CMDLIST) and the Control
316 * List (CTL).
317 *
318 CMDLIST   DB    #$03        ;Number of items in CMDLIST
319 P4        DB    DRV         ;Drive # requested by user
320 P5        DA    CTL         ;Address of Control List
321 P6        DB    SUBCMD      ;Subcommand
322 CTL       DB    SIZE        ;Number of items in CTL if doing
323 *                            the eject.  Device Status Byte
324 *                            if doing the 'GET DSB'.
325 *
326 * Here are the other 24 bytes in the CTL.
327 *
328           DB    $00         ;Size (Lo-byte)
329           DB    $00         ;Size (Middle-byte)
330           DB    $00         ;Size (Hi-byte)
331           DB    $00         ;ID string length
332           DS    16,32       ;ID string (32='SPACE')
333 P7        DB    TYPE        ;Type byte for device
334           DB    $00         ;Subtype for device
335           DW    $0000       ;Version (word)
336 *
337 * That's it!
338 *
```

END OF LISTING 1

LISTING 2: DISK.EJECTOR

```
Start: 8000 Length: 115

70 8000:A5 50 85 FD A5 51 85 FE
05 8008:18 A5 FD 69 72 85 FD A5
B7 8010:FE 69 00 85 FE 20 BE DE
96 8018:20 67 DD 20 52 E7 A5 50
D1 8020:C9 08 90 05 A9 CB 85 FF
ED 8028:60 69 C0 85 FC A9 00 85
04 8030:FB 20 BE DE 20 67 DD 20
29 8038:52 E7 A5 50 85 FA A0 01
4F 8040:B1 FB C9 20 F0 05 A9 CC
63 8048:85 FF 60 A0 03 B1 FB C9
EB 8050:00 F0 05 A9 CC 85 FF 60
7D 8058:A0 05 B1 FB C9 03 F0 05
45 8060:A9 CC 85 FF 60 A0 07 B1
35 8068:FB C9 00 F0 05 A9 CC 85
D9 8070:FF 60 A0 FF B1 FB 18 69
FE 8078:03 85 FB A0 7D 91 FD A5
87 8080:FC C8 91 FD A5 FD A0 80
15 8088:18 69 85 91 FD A5 FE 69
87 8090:00 C8 91 FD A5 FD A0 87
F0 8098:18 69 8A 91 FD A5 FE 69
E9 80A0:00 C8 91 FD A0 55 A5 FD
6F 80A8:18 69 7C 91 FD A5 FE 69
38 80B0:00 C8 91 FD A5 FA A0 86
3D 80B8:91 FD A9 00 A0 7F 91 FD
83 80C0:A9 03 A0 89 91 FD 20 EE
F2 80C8:80 90 03 85 FF 60 A0 9F
95 80D0:B1 FD C9 01 F0 05 A9 CA
96 80D8:85 FF 60 A9 04 A0 7F 91
31 80E0:FD A0 89 91 FD A9 00 A0
CE 80E8:8A 91 FD C8 91 FD 20 00
07 80F0:00 00 F7 80 85 FF 60 03
B4 80F8:00 FC 80 00 00 00 00 00
8D 8100:00 20 20 20 20 20 20 20
A0 8108:20 20 20 20 20 20 20 20
3C 8110:20 00 00 00 00

TOTAL: 6552
```

END OF LISTING 2

LISTING 3: EJECT.DEMO

```
37 | 10  REM **********************
C0 | 20  REM *      EJECT.DEMO      *
B9 | 30  REM *   BY TIM SWIHART    *
AE | 40  REM * COPYRIGHT (C) 1988 *
CB | 50  REM * BY MICROSPARC, INC *
24 | 60  REM * CONCORD, MA  01742 *
45 | 70  REM **********************
50 | 80  ONERR  GOTO 180
5E | 90 S = 5:D = 1: REM  SLOT 5, DRIVE 1
4E | 100 D$ =  CHR$ (4):EJ = 32768
1C | 110  PRINT D$;"BLOAD DISK.EJECTOR,A$8000"
6A | 120  TEXT : HOME : PRINT "ATTEMPTING TO EJECT F
   |      ROM SLOT ";S
49 | 130  PRINT  SPC( 24)"DRIVE ";D
A0 | 140  CALL EJ,S,D
35 | 150  IF  PEEK (255) <  > 0 THEN 180
3F | 160  PRINT : PRINT "SUCCESS!"; CHR$ (7)
85 | 170  END
4D | 180 ER =  PEEK (255):EC =  PEEK (222):EL =  PEE
   |      K (218) + 256 *  PEEK (219)
F7 | 190  IF ER = 39 THEN  PRINT : PRINT "I/O ERROR"
   |      ; CHR$ (7): END
2A | 200  IF ER = 40 THEN  PRINT : PRINT "NO DEVICE
   |      CONNECTED"; CHR$ (7): END
4B | 210  IF ER = 202 THEN  PRINT : PRINT "REQUESTED
   |       DRIVE IS NOT A 3.5"; CHR$ (34);" DRIVE.";
   |      CHR$ (7): END
09 | 220  IF ((ER = 203) AND (EC = 16)) THEN  PRINT
   |      : PRINT "REQUESTED SLOT NUMBER MUST": PRINT
   |      "BE IN RANGE 0 TO 7."; CHR$ (7): END
9F | 230  IF ER = 204 THEN  PRINT : PRINT "REQUESTED
   |      SLOT IS NOT A SMARTPORT"; CHR$ (7): END
B1 | 240  IF EC = 16 THEN  PRINT : PRINT "SYNTAX ERR
   |      OR IN LINE ";EL; CHR$ (7): END
42 | 250  PRINT : PRINT "ERROR NUMBER ";ER;" HAS OCC
   |      URRED IN LINE ";EL; CHR$ (7): END

TOTAL: AA4B
```

END OF LISTING 3