APPLE
UTILITIES

PROGRAMMING THE APPLEMOUSE II

DOS 3.3 ○ 0

Add a mouse to your Apple II series computer, then use the techniques described here to create programs that use it. Both Applesoft and assembly language programming techniques are described and illustrated with two sample programs.

ProDOS ○ 0

by Sandy Mossberg, 50 Talcott Rd., Rye Brook, NY 10573

Mouse technology offers exciting prospects to owners of Apple II series computers. You no longer need a 16- or 32-bit computer to produce pull-down menus, icons and sophisticated graphics. Simply amble into your friendly neighborhood computer pet shoppe, plunk down $100 to $150 or a plastic card, and take the creature home.

CONNECTING THE BEAST

The Apple //c contains built-in mouse firmware. The other II series Apples require a card that can be plugged into any vacant expansion slot except slot zero on the II/II Plus and slot 3 on the //e with an 80-column card in the auxiliary slot. Slot 4 is recommended.

On the //c, hooking up the mouse is as simple as plugging the cable connector into the mouse/joystick port on the back of your computer. Installing the //e, II or II Plus mouse, especially assembling the connector, demands a high degree of eye-hand coordination. It would be a snap for a brain surgeon — I took about 30 minutes, but then again I often poke myself in the eye while attempting to scratch my forehead.

MOUSEPAINT DRAWING PROGRAM

Your first introduction to Apple II mouse power is the MousePaint disk that comes with the package. This remarkable program is Bill Budge's adaptation of MacPaint for Apple II series computers. Although the graphics are not as crisp as on the Macintosh and it lacks some of the bells and whistles, the program is a winner. Expect to find pull-down menus (File, Edit, Aids and Fonts), pattern boxes, variable line widths, enclosed shapes (solid and hollow), and drawing tools such as the pencil, spray can, brush, straight edge, text letter and eraser. The familiar grabbing hand and editor's box are also present. The paint can and lasso are missing, but this detracts little from the program. All considered, I predict that you'll love MousePaint.

DOCUMENTATION

The *AppleMouse II User's Manual For the Apple //e, II Plus and II* is similar to the *AppleMouse //c User's Manual*. The former publication goes into confusing detail about assembling the connector on various flavors of Apple, while the latter manual simply shows a picture of how to plug in the connector. When it comes to hardware, simpler is better. MousePaint is described adequately in both manuals. In many respects, this section is superior to the dismal documentation of MacPaint.

The care and feeding of the mouse are handled in a cavalier fashion. Too much attention is given to the mouse's "tummy." The section on dissecting the mouse is quite distasteful, even for a vivisectionist. The final blow is the admonition not to let the rodent "run through wet or oily spots, dust, grit or cookie crumbs." How in blazes is the creature going to survive if we starve it?

Both manuals contain an adequate chapter on programming the mouse in BASIC. For some inexplicable reason, however, the important chapter on mouse firmware is omitted from the //c manual. Sure, peripheral cards are only included with the //e, II, and II Plus kits, but the //c contains the same firmware. The information provided in this section is essential for assembly language (A.L.) and advanced BASIC programmers. Wise up Cupertino! — //c owners are first-class citizens. I'll cover for you this time (in this article), but from here on, Apple, you're on your own!

MOUSE PROGRAMMING

Programs for the mouse function under the DOS 3.3 or ProDOS environment. The subsequent material should provide you with the principal features of writing mouse programs in BASIC and A.L. Both sample programs function on all Apple II series computers using either major operating system.

TABLE 1: Mode Byte Attributes

Mode Bit	Function (if set)
0	Turns the mouse on
1	Enables interrupt on mouse movement
2	Enables interrupt when the mouse button is down
3	Enables interrupt on each screen refresh cycle
4-7	Reserved (must be zero)

FINDING THE MOUSE

If a mouse card occupies a peripheral slot, the following two locations (in which *n* equals the slot number) contain values that identify the firmware as belonging to a mouse:

Address	Contents
$C*n*0C	$20
$C*n*FB	$D6

To locate the mouse, simply scan each expansion slot for these two signature bytes. This technique will be described later for BASIC and A.L. programs.

Although I encourage you to write programs that function on all Apple II series computers, those who author dedicated //c software can be assured that Mr. Mouse lives in hole number 4. Thus, location $C40C (50188) contains $20 (32), and $C4FB (50427) holds $D6 (214).

PROGRAMMING THE MOUSE IN BASIC

The mouse functions like any other peripheral device. For illustrative purposes, we shall assume that it is in slot 4.

Turning On the Mouse

To awaken the mouse, you must nudge it with ASCII character 1. The following program line does the trick:

```
PRINT CHR$(4)“PR#4” : PRINT CHR$(1)
```

The first statement assigns output to slot 4, and the second statement activates the mouse with its favorite cheese, ASCII character 1.

Once the mouse has been turned on, output may be routed to the screen by the command **PRINT CHR$(4)“PR#0”.**

Communicating With the Mouse

The mouse's position and button status can be determined by the following program line:

```
PRINT CHR$(4)“IN#4” : INPUT “ ”; X,Y,S
```

The first command assigns input to slot 4, and the second command places data into the three listed variables. X contains the horizontal position of the mouse, Y holds the vertical coordinate, and S specifies the button status. The empty quotation marks suppress printing of the question mark prompt evoked by the plain INPUT command.

The X,Y coordinates range from zero to 1,023. With the mouse's tail pointed away from you, X increases with movement to your right and Y increases with motion toward you.

The status variable holds a value of −4 to +4. A negative number indicates that a key has been pressed, in which case S will remain negative until the keyboard strobe is reset with the command POKE −16368,0. The following table translates the possible values (positive or negative) for S (where P indicates the button is pressed and R indicates the button has been released):

S	Current	Prior
1	P	P
2	P	R
3	R	P
4	R	R

To receive input from the keyboard, enter the command **PRINT CHR$(4)“IN#0”**. If you need to poll the mouse again, remember to re-establish input from slot 4.

Turning Off the Mouse

The mouse is deactivated by sending it an ASCII character 0, as illustrated below:

```
PRINT CHR$(4)“PR#4” : PRINT CHR$(0)
```

TABLE 2: READMOUSE Transfers

Screen Hole Address	Content
$478 + *n*	Low byte of the X-coordinate
$4F8 + *n*	Low byte of the Y-coordinate
$578 + *n*	High byte of the X-coordinate
$5F8 + *n*	High byte of the Y-coordinate
$678 + *n*	Reserved
$6F8 + *n*	Reserved
$778 + *n*	Button and interrupt status
$7F8 + *n*	Current mode

The first command assigns output to slot 4, and the second command deactivates the mouse.

BASIC Demo Program: Lo-Res MouseSketch

Both mouse manuals contain MOUSE.DRAW, a short demonstration program. MOUSE.SKETCH (**Listing 1**) expands on the MOUSE.DRAW theme to cover a full range of BASIC mouse manipulations.

MOUSE.SKETCH enables you to produce line drawings on the low resolution (Lo-Res) screen using your mouse. Your position on the screen is indicated by a mouse cursor. The screen location may be filled with a white color by pressing the mouse button. A filled box may be erased by pressing the open-apple or closed-apple key (equivalent to the paddle buttons on the II and II Plus) in conjunction with the mouse button. Pressing <CTRL>C clears the screen, and <ESC> ends the sketching session.

To key in MOUSE.SKETCH, type in the program as shown in **Listing 1** and save it with the command:

```
SAVE MOUSE.SKETCH
```

For help in entering *Nibble* listings, see ''A Welcome to New *Nibble* Readers'' at the beginning of this issue.

MOUSE.SKETCH is well annotated. Important program variables are shown at the beginning of the listing. **Line 210** calls the subroutine that locates mouse firmware (**lines 710-770**). Starting with slot 1, successive slots are searched for the correct identification bytes. If the appropriate firmware is located, the slot number is assigned to N and the subroutine returns. If no firmware is found, the return address is popped from the stack, a message is printed, and the program ends.

Line 220 calls the subroutine (**lines 620-670**) that sets mixed Lo-Res and text mode, awakens the mouse and directs output to the screen. Since the blank screen is clear (black), the color of the current screen coordinate (C) is set to black.

Line 230 directs input to be obtained from the mouse port, and **line 270** calls the subroutine (**lines 390-420**) that reads the mouse

TABLE 3: Button and Interrupt Status (BIS) Byte Attributes

BIS Bit	Meaning (if set)
0	Reserved
1	Interrupt caused by mouse movement
2	Interrupt caused by button down
3	Interrupt caused by screen refresh
4	Reserved
5	X or Y changed since prior reading
6	Button down at prior reading
7	Button down currently

position and button data. The 20-row Lo-Res screen is a 40 × 40 grid. **Lines 400-410** convert raw position values into Lo-Res coordinates (the number 25.575 is obtained by dividing 1,023 by 40).

If the mouse is stationary and no event (e.g., button down, keypress or mouse movement) has occurred, **lines 320-330** put the cursor on the screen. On a color monitor, the cursor is magenta; on a monochrome monitor it appears as a hatched box. **Line 340** assigns current X,Y values to OX and OY so that a change in position can later be documented. **Line 350** loops back for another data poll.

Line 280 tests for the down position of either apple key. If an apple key and the mouse button are pressed together, the current Lo-Res coordinate is colored black, i.e., an unfilled (black) box remains black, whereas a filled (white) box is erased (made black).

Line 290 tests for mouse movement by comparing OX to X and OY to Y. If the mouse position has changed and the mouse button is or was up, the old cursor is cleared, the color of the new screen coordinate is read and placed into C, and flow branches to the lines that produce the cursor.

Line 300 checks for a keypress, i.e., a negative value for S. If a key is down, control passes to **lines 460-510** where the keyboard strobe is reset, input is accepted from the keyboard rather than the mouse, a message is printed on the text portion of the screen, and input is solicited. <CTRL>C clears the sketching screen, <RETURN> returns you to the current sketching screen, and <ESC> ends the program. On termination (**lines 550-580**), full text mode is set, the mouse is deactivated, and output is routed to the screen.

> To awaken the mouse, you must nudge it with ASCII character 1.

When you come to understand this BASIC code, you'll be well on your way to becoming a competent mouse programmer. You might even wish to enhance MOUSE.SKETCH by adding a command that saves sketches to disk. Don't you agree that, aside from being new and different, mouse programming is great fun?

PROGRAMMING THE MOUSE WITH ASSEMBLY LANGUAGE

The A.L. programmer interacts with expansion slot firmware by accessing three special areas of memory:

1. Peripheral Card ROM Space is a 256-byte area ($C*n*00 to $C*n*FF, where *n* is the slot number). Simply plugging a card into an expansion slot fills this space with binary code.
2. Peripheral Card I/O Space occupies the 16 bytes $C080 + Y to $C08F + Y, where Y equals the slot number times 16. These *device select* software switches allow direct communication with the peripheral firmware ROM. Although these switches may be used directly by the A.L. programmer, they are usually referenced by the code in the Peripheral Card ROM Space.
3. Peripheral Slot Scratchpad RAM consists of eight locations for each expansion slot (1-7) and is used primarily to store data. Because these addresses fall within the text and Lo-Res video display (but their contents do not appear on the screen and are not affected by normal screen operations), they are called *screen holes* and will be considered later in greater detail.

TABLE 4: Clamping Values

Screen Hole Address	Content
$478	Low byte of the X-coordinate
$4F8	Low byte of the Y-coordinate
$578	High byte of the X-coordinate
$5F8	High byte of the Y-coordinate

TABLE 5
Low-Order Addresses for Mouse Firmware Routines

Address	Routine
$Cn12	SETMOUSE
$Cn13	SERVEMOUSE
$Cn14	READMOUSE
$Cn15	CLEARMOUSE
$Cn16	POSMOUSE
$Cn17	CLAMPMOUSE
$Cn18	HOMEMOUSE
$Cn19	INITMOUSE

Although the above description is generic, it holds true for mouse firmware. We shall now review how A.L. programs can control the mouse.

Mouse Modes

Passive mode represents the simplest way to manage the mouse. All functions are performed within the firmware without disturbing the main system.

In interrupt mode the mouse firmware sends an interrupt (IRQ) signal to the Apple's central processing unit whenever a valid interrupt event occurs. Generally, the interrupt is serviced during the monitor's vertical refresh cycle.

The mode is set during the SETMOUSE call described in the next section. The low-order nibble of the mode byte contains all the pertinent information, as shown in **Table 1**.

Mouse Routines

Eight firmware routines are available to manipulate the mouse:

1. INITMOUSE sets the internal default values for mouse firmware and synchronizes its function with the vertical blanking cycle. This routine must be invoked prior to any other mouse routine and is best called before a screen display is created.
2. SETMOUSE starts or stops mouse operation, depending upon the mode byte contents in the A-Register. If the Accumulator contains zero, the mouse is disabled. An A-Register value of 1 sets passive mode. Values of $2-$F set interrupt mode.
3. READMOUSE transfers mouse data from the firmware to the screen holes as listed in **Table 2** (where *n* equals the slot number). The attributes of the button and interrupt status (BIS) byte are given in **Table 3**. READMOUSE clears bits 1-3 in the BIS byte. Mouse movement can be measured over a maximal range of 65,536 units; however, default values are restricted to a range of 0-1,023.
4. CLEARMOUSE zeros the X,Y coordinates, both on the firmware and in the screen holes. The BIS byte remains intact.
5. SERVEMOUSE updates the BIS byte to reveal which event caused the interrupt. Screen holes remain unchanged. On exit, a clear Carry indicates the interrupt was caused by the mouse, whereas a set Carry flags a non-mouse interrupt.

6. CLAMPMOUSE sets new values for mouse position data in accord with the contents of the screen hole locations listed in **Table 4**. If the A-Register contains a zero, CLAMPMOUSE sets the X-coordinate range. If the Accumulator is nonzero, the Y-coordinate range is clamped. This routine scrambles the contents of the X,Y position screen holes (they may be restored with READMOUSE).
7. HOMEMOUSE sets the firmware position data to the lowest values permitted. This call is equivalent to setting the mouse position to the upper-left corner of the clamping window. The screen hole values remain intact (they may be reset with READMOUSE).
8. POSMOUSE sets the firmware position registers to the values in the X,Y position screen holes.

Calling the Mouse

The entry point addresses for each of the mouse routines are contained within a table in the firmware and can be derived in the following manner. The high-order byte is always C*n*, where *n* is the slot number. The look-up table (**Table 5**) provides only the low-order address for each routine. For example, if the mouse lives in slot 4, the entry point to set the mouse is calculated by adding the content of location $C412 to the value $C400. One way of doing this is described in the demonstration program that follows.

Before the actual mouse call is made, the X- and Y-Registers must contain the C*n* value (e.g., $C4 for slot 4) and the *n*0 value (slot number times $10, e.g., $40 for slot 4). Except for SERVEMOUSE, the Carry bit indicates whether the call was completed successfully (on Carry Set, an error has occurred).

A.L. Demo Program MOUSE.TRACK

MOUSE.TRACK (**Listing 2**) is more complex than the sample A.L. program in the //e manual. A mouse cursor is placed on the screen, and a status line provides the X,Y coordinates of the cursor and the bit values of the BIS byte. Pressing any key ends the program. It may not sound very exciting, but the techniques employed will give you a head start as an A.L. mouse programmer.

Use an assembler to enter the source code as shown in **Listing 2**, or use the Monitor to enter the code directly. Save the program with the command:

BSAVE MOUSE.TRACK,A$6000,L$1F6

After setting a normal text window (**line 47**), a call (**line 48**) to the CHKMOUSE subroutine (**lines 252-295**) searches the slot firmware for the mouse ID bytes. If the beast is not located, a message is printed and the program ends. On finding mouse firmware, the storage locations N, CN and N0 (**lines 299-301**) are filled with the slot number, C*n* value, and the slot number times $10, respectively. In case forty-column mode is not active, this is accomplished by outputting <CTRL>Q via COUT (**lines 49-50**).

Calls are handled by the CALLFIRM subroutine (**lines 201-207**), which preserves the entry A-Register and loads the X- and Y-Registers with CN and N0, respectively. Self-modifying code (**lines 202-203**) produces the correct jump instruction (**line 207**).

After awakening the mouse with INITMOUSE (**lines 51-52**), the screen is formatted (**line 53**) and SETMOUSE starts the mouse in passive mode (**lines 54-56**).

Because we shall be tracking the mouse's position on a screen containing 40 columns and 24 rows, it makes sense to change the default X-and Y-ranges from 0-1023 to values that make plotting of screen coordinates easier. Since 40 times 24 equals 960, a clamping window of 0-959 is set by CLAMPMOUSE for both axes (**lines 57-64**). After homing the mouse (**lines 65-66**) and guaranteeing a clear keyboard strobe (**line 67**), mouse tracking begins in earnest.

Following a set-up call to READMOUSE (**lines 71-72**), control passes to **line 83**, where the cursor position is set by calling SETPOSN (**lines 108-135**). Here, the screen hole data is extracted and values for CH and CV are fixed. Note that each of the 40 columns is represented by 24 movement units (40 × 24 = 960), whereas each of the 24 rows requires 40 movement units (24 × 40 = 960). You may wonder why I did not choose to set the clamping window to a range of 0-23 for the Y-axis and 0-39 for the X-coordinates. The answer is straightforward — such low clamping values would magnify mouse movement such that only minimal motion would advance the cursor across the entire screen. When you become comfortable with this program, try these small clamping values, alter SETPOSN to reflect the new range, and observe this phenomenon firsthand. You will probably want to lower the clamping values when you apply these techniques to a real program. The current values offer extremely high resolution, but the mouse requires a very large operating surface.

The loop formed by **lines 74-89** continually updates mouse data. **Lines 74-75** read the firmware and **line 76** calls the subroutine that prints the information on the screen. Mouse motion is detected by testing bit 5 of the BIS byte (**lines 77 and 79**). If the bit is clear (**line 80**), the mouse has been stationary and flow passes to **lines 86-88**, which put the cursor on the screen and check for a keypress. If bit 5 of the BIS byte is set, the mouse has been scurrying about, in which case the cursor is replaced with the screen character that formerly occupied that position (**lines 81-82**), the new position is calculated (**line 83**), and the screen character at that location is saved (**lines 84-85**) before the cursor is printed (**lines 86-87**).

If a key is pressed, the branch in **line 89** is not taken and flow falls to the exit code. **Line 93** resets the keyboard strobe, **lines 94-95** obliterate the cursor, **lines 96-98** turn off the mouse, and **lines 99-102** exit to Applesoft.

That's not difficult at all. At the risk of repeating myself, mouse programming is fun.

THE CRYSTAL BALL

By the time this article is in print, two important new products will be available for your Apple. The *Apple //e Enhancement Kit* turns your //e into a more potent tool. Four replacement chips include a 65C02 CPU with its enhanced instruction set and faster processing, a character generator that provides graphic icons, and two Monitor ROM chips. The new Monitor allows lower-case Applesoft commands, includes a mini-assembler, and has an ASCII search capability. Interrupts are supported. Thus, for a nominal price, your old //e may be converted to a more powerful //c-like machine without losing its own personality and expandability.

The *MouseText Tool Kit* provides a Macintosh-like environment in the //c and in the //e that has been updated with the Enhancement Kit. The heart of the kit is a set of machine language routines that can be accessed either from A.L. or from BASIC via the ampersand command. ProDOS is required. The speed of the BASIC interface approaches that of a binary program. My experience with a pre-release version of the Tool Kit indicates that amazing capabilities are in store for Apple II series enthusiasts.

COMMENTARY

I enjoy taking a swipe at Apple Computer as well as the next guy, but it deserves praise too. I have great admiration for a company that continues to support purchasers of older equipment. The Apple II series has evolved from the plain vanilla Apple II to the II Plus, //e and //c. Yet the newest computer runs much of the home-brewed software written for the oldest model. I can run a DOS 3.2 Integer BASIC program on my //c or enhanced //e. That's remarkable. Thank you, Apple. (Now, just bring down the price of the Macintosh 512K expansion board and I'll be your biggest fan!)

LISTING 1: MOUSE.SKETCH

```
1   REM **********************
2   REM *    MOUSE.SKETCH    *
3   REM * BY SANDY MOSSBERG  *
4   REM * COPYRIGHT (C) 1985 *
5   REM * BY MICROSPARC, INC *
6   REM * CONCORD, MA  Ø1742 *
7   REM **********************
1ØØ  REM *********************************
11Ø  REM *  VARIABLE USAGE:              *
12Ø  REM *     X = HORIZONTAL COORDINATE *
13Ø  REM *     Y = VERTICAL COORDINATE   *
14Ø  REM *    OX = PRIOR X VALUE         *
15Ø  REM *    OY = PRIOR Y VALUE         *
16Ø  REM *     C = COLOR AT X,Y          *
17Ø  REM *     S = STATUS OF MOUSE BUTTON *
18Ø  REM *     N = SLOT OF MOUSE FIRMWARE *
19Ø  REM *********************************
2ØØ  :
21Ø  GOSUB 71Ø: REM TEST FOR MOUSE FIRMWARE
22Ø  GOSUB 62Ø: REM INITIALIZE
23Ø  PRINT D$"IN#"N: REM GET INPUT FROM MOUSE
24Ø  REM ===================
25Ø  REM TRACK PATH OF MOUSE:
26Ø  REM ===================
27Ø  GOSUB 39Ø: REM GET MOUSE POSITION DATA
28Ø  IF  PEEK (49249) >  = 128 OR  PEEK (4925
     Ø) >  = 128 THEN  IF S < 3 THEN C = Ø: COLOR=
     Ø: GOTO 33Ø: REM IF MOUSE BUTTON DOWN AN
     D OPEN/CLOSED-APPLE PRESSED, CLEAR POINT
      ON SCREEN (SET COLOR TO BLACK)
29Ø  IF OX <  > X OR OY <  > Y THEN  IF S >  =
     2 THEN  COLOR= C: PLOT OX,OY:C =  SCRN(
     X,Y): REM IF MOUSE POSITION HAS CHANGED,
      CLEAR PRIOR CURSOR AND READ NEW SCREEN
     COORDINATE
3ØØ  IF S < Ø THEN 46Ø: REM PROCESS KEYPRESS
31Ø  IF S <  = 2 THEN  COLOR= 15:C = 15: GOTO
     33Ø: REM IF MOUSE BUTTON DOWN, SET COLOR
      TO WHITE
32Ø  COLOR= 1: REM CURSOR COLOR IS MAGENTA (H
     ATCHED BOX)
33Ø  PLOT X,Y: REM PUT COLOR ON SCREEN
34Ø OX = X:OY = Y: REM CURRENT COORDINATES NO
     W OLD HAT
35Ø  GOTO 27Ø: REM LOOP BACK FOR MORE INPUT
36Ø  REM ==================
37Ø  REM OBTAIN MOUSE INPUT:
38Ø  REM ==================
39Ø  INPUT "";X,Y,S: REM READ MOUSE DATA
4ØØ X =  INT (X / 25.575): REM CONVERT MOUSE
     POSITION HORIZONTAL COORDINATES (Ø-1Ø23)
      TO LORES COORDINATES (Ø-4Ø)
41Ø Y =  INT (Y / 25.575): REM SAME FOR VERTI
     CAL COORDINATES
42Ø  RETURN
43Ø  REM ====================
44Ø  REM CHECK KEYBOARD INPUT:
45Ø  REM ====================
46Ø  POKE  - 16368,Ø: REM CLEAR KEYBOARD STRO
     BE
47Ø  PRINT D$"IN#Ø": REM ACCEPT INPUT FROM KE
     YBOARD
48Ø  VTAB 22: PRINT "PRESS RETURN TO CONTINUE
     , ESC TO QUIT OR CTL-C TO CLEAR SCREEN "
     ;: GET A$: REM PROMPT TO CONTINUE, QUIT
     OR CLEAR SCREEN
49Ø  PRINT : IF A$ =  CHR$ (3) THEN 22Ø: REM
      CLEAR SCREEN IF CTL-C PRESSED
5ØØ  IF A$ =  CHR$ (13) THEN  HOME : PRINT D$
     "IN#"N: GOTO 27Ø: REM CONTINUE IF RETURN
      PRESSED
51Ø  IF A$ <  >  CHR$ (27) THEN  PRINT  CHR$
     (7): GOTO 48Ø: REM TRAP ERRONEOUS KEYPRE
     SS
52Ø  REM =====
53Ø  REM QUIT:
54Ø  REM =====
55Ø  TEXT : HOME
56Ø  PRINT D$"PR#"N: PRINT  CHR$ (Ø): REM DEA
     CTIVATE MOUSE
57Ø  PRINT D$"PR#Ø": REM SEND OUTPUT TO SCREE
     N
58Ø  PRINT "THE MOUSE IS SLEEPING...": END
59Ø  REM ===========================
6ØØ  REM INITIALIZE SCREEN AND MOUSE:
61Ø  REM ===========================
62Ø  HOME : GR : REM CLEAR SCREEN AND SET LOR
     ES
63Ø D$ =  CHR$ (4): REM DEFINE DOS STRING. FO
     R DOS 3.3 USE D$=CHR$(13)+CHR$(4)
64Ø C = Ø: REM STARTING POINT BLANK
65Ø  PRINT D$"PR#"N: PRINT  CHR$ (1): REM ACT
     IVATE MOUSE
66Ø  PRINT D$"PR#Ø": REM SEND OUTPUT TO SCREE
     N
67Ø  RETURN
68Ø  REM =========================
69Ø  REM SEARCH FOR MOUSE FIRMWARE:
7ØØ  REM =========================
71Ø L1 = 4942Ø:L2 = 49659: REM START WITH SLO
     T 1 MOUSE FIRMWARE ID BYTES (L1=$C1ØC, L
     2=$C1FB)
72Ø  FOR I = 1 TO 7: REM TEST SLOTS 1-7
73Ø  IF  PEEK (L1) = 32 AND  PEEK (L2) = 214 THEN
     N = I:I = 9: REM IF MOUSE FIRMWARE LOCAT
     ED, N=SLOT # AND I > 8 FLAGS THE MATCH
74Ø L1 = L1 + 256:L2 = L2 + 256: REM SET FOR
     NEXT HIGHER SLOT
75Ø  NEXT I
76Ø  IF I > 8 THEN  RETURN : REM MOUSE FIRMWA
     RE FOUND
77Ø  POP : PRINT  CHR$ (7);: PRINT "MOUSE FIR
     MWARE NOT FOUND...": REM MOUSE FIRMWARE
     NOT LOCATED
```

END OF LISTING 1

```
          KEY PERFECT 4.Ø
               RUN ON
            MOUSE.SKETCH
    ============================
       CODE       LINE# - LINE#
       ------     -------------
       6A89          1  -   12Ø
       6CDF        13Ø  -   22Ø
       F1ØD        23Ø  -   32Ø
       A21F        33Ø  -   42Ø
       C855        43Ø  -   52Ø
       76F4        53Ø  -   62Ø
       A37E        63Ø  -   72Ø
       7Ø6B        73Ø  -   77Ø
     PROGRAM CHECK IS : ØAØ9
```

LISTING 2: MOUSE.TRACK

```
        *********************
        * MOUSE.TRACK       *
        * BY SANDY MOSSBERG *
        * COPYRIGHT (C) 1985 *
        * BY MICROSPARC, INC *
        * CONCORD, MA  Ø1742 *
        *********************
        * Merlin Assembler

        * General Equates:

PTR       =    $Ø6       ;Pointer, temp storage
CH        =    $24       ;Column
CV        =    $25       ;Row
BASL      =    $28       ;Left char of current row
DOSWARM   =    $3DØ      ;Warm-start (Pro)DOS
KBD       =    $CØØØ     ;Keyboard input
STROBE    =    $CØ1Ø     ;Keyboard strobe
LINPRT    =    $ED24     ;Print decimal of A,X
PRBLNK    =    $F948     ;Print 3 blanks
TEXT      =    $FB39     ;Set normal text window
TABV      =    $FB5B     ;Set row in A-reg
HOME      =    $FC58     ;Home cursor, clear screen
CROUT     =    $FD8E     ;Output CR
COUT      =    $FDED     ;Output char

        * Screenhole Equates:

XL        =    $478      ;+n=lo byte X-position
YL        =    $4F8      ;+n=lo byte Y-position
XH        =    $578      ;+n=hi byte X-position
YH        =    $5F8      ;+n=hi byte Y-position
```

```
                  BUTTON   =      $778          ;+n=button status

                  * Offsets to Mouse Entry Points:

                  SETMSE   =      $12
                  READMSE  =      $14
                  CLAMPMSE =      $17
                  HOMEMSE  =      $18
                  INITMSE  =      $19

                           ORG    $6000
                  *-------------------------------
                  * Initialize:
                  *-------------------------------
6000: 20 39 FB             JSR    TEXT          ;Set text mode
6003: 20 93 61             JSR    CHKMOUSE      ;Check for Mouse firmware
6006: A9 91                LDA    #$91          ;CTL-Q
6008: 20 ED FD             JSR    COUT          ;Set 40 columns
600B: A0 19                LDY    #INITMSE
600D: 20 17 61             JSR    CALLFIRM      ;Initialize Mouse firmware
6010: 20 27 61             JSR    FMTSCR        ;Format screen
6013: A0 12                LDY    #SETMSE
6015: A9 01                LDA    #1            ;Set passive mode
6017: 20 17 61             JSR    CALLFIRM      ;Start mouse
601A: A0 17                LDY    #CLAMPMSE
601C: 20 B0 60             JSR    SETCLAMP      ;Set new clamping values
601F: A9 00                LDA    #0            ; for X-coordinate
6021: 20 17 61             JSR    CALLFIRM      ;Clamp X-coordinate
6024: A0 17                LDY    #CLAMPMSE
6026: 20 B0 60             JSR    SETCLAMP      ;Set new clamping values
6029: A9 01                LDA    #1            ; for Y-coordinate
602B: 20 17 61             JSR    CALLFIRM      ;Clamp Y-coordinate
602E: A0 18                LDY    #HOMEMSE
6030: 20 17 61             JSR    CALLFIRM      ;Home Mouse position
6033: 2C 10 C0             BIT    STROBE        ;Reset keyboard strobe
                  *-------------------------------
                  * Track the Creature:
                  *-------------------------------
6036: A0 14       TRAKMOUS LDY    #READMSE
6038: 20 17 61             JSR    CALLFIRM      ;Read initial position
603B: 90 16                BCC    :2            ;Set initial cursor (always)
603D: A0 14       :1       LDY    #READMSE
603F: 20 17 61             JSR    CALLFIRM      ;Read Mouse position
6042: 20 C3 60             JSR    PRTDATA       ;Print data to screen
6045: B9 78 07             LDA    BUTTON,Y      ;Get Mouse button status
6048: A4 24                LDY    CH
604A: 29 20                AND    #%00100000    ;Test bit 5
604C: F0 0D                BEQ    :3            ;X,Y unchanged
604E: AD F9 61             LDA    OLDCHAR       ;X,Y changed so
6051: 91 28                STA    (BASL),Y      ; restore screen char
6053: 20 7E 60    :2       JSR    SETPOSN       ;Set cursor position
6056: B1 28                LDA    (BASL),Y
6058: 8D F9 61             STA    OLDCHAR       ;Save screen char
605B: A9 DE       :3       LDA    #"^"
605D: 91 28                STA    (BASL),Y      ;Print cursor
605F: 2C 00 C0             BIT    KBD           ;Check keypress
6062: 10 D9                BPL    :1            ;No keypress. Loop back
                  *-------------------------------
                  * Quit:
                  *-------------------------------
6064: 2C 10 C0             BIT    STROBE        ;Reset keyboard strobe
6067: AD F9 61             LDA    OLDCHAR
606A: 91 28                STA    (BASL),Y      ;Kill cursor
606C: A0 12                LDY    #SETMSE
606E: A9 00                LDA    #0
6070: 20 17 61             JSR    CALLFIRM      ;Turn Mouse off
6073: A9 04                LDA    #4
6075: 20 5B FB             JSR    TABV
6078: 20 8E FD             JSR    CROUT
607B: 4C D0 03             JMP    DOSWARM       ;Exit to Applesoft
                  *-------------------------------
                  * Set Cursor Position:
                  *-------------------------------
                  * Set row:

607E: AE F6 61    SETPOSN  LDX    N
6081: BD F8 05             LDA    YH,X
6084: 85 08                STA    PTR+2
6086: A0 FF                LDY    #-1
6088: BD F8 04             LDA    YL,X
608B: 38          :4       SEC
608C: E9 28       :5       SBC    #40           ;Y-units per row
608E: C8                   INY
608F: B0 FB                BCS    :5
6091: C6 08                DEC    PTR+2
6093: 10 F6                BPL    :4
6095: 98                   TYA
6096: 20 5B FB             JSR    TABV

                  * Set column:

6099: BD 78 05             LDA    XH,X
609C: 85 08                STA    PTR+2
609E: A0 FF                LDY    #-1
60A0: BD 78 04             LDA    XL,X
60A3: 38          :6       SEC
60A4: E9 18       :7       SBC    #24           ;X-units per column
60A6: C8                   INY
60A7: B0 FB                BCS    :7
60A9: C6 08                DEC    PTR+2
60AB: 10 F6                BPL    :6
60AD: 84 24                STY    CH
60AF: 60                   RTS
                  *-------------------------------
                  * Set New Clamping Values:
                  *-------------------------------
                  * Entry conditions:
                  *     XL/H = lo boundary
                  *     YL/H = hi boundary
                  *
60B0: A9 00       SETCLAMP LDA    #0            ;Min=0
60B2: 8D 78 04             STA    XL
60B5: 8D 78 05             STA    XH
60B8: A9 BF                LDA    #$BF          ;Max=959 ($3BF)
60BA: 8D F8 04             STA    YL
60BD: A9 03                LDA    #3
60BF: 8D F8 05             STA    YH
60C2: 60                   RTS
                  *-------------------------------
                  * Print Data Line to Screen:
                  *-------------------------------
60C3: A5 25       PRTDATA  LDA    CV
60C5: 48                   PHA                  ;Save entry row
60C6: A5 24                LDA    CH
60C8: 48                   PHA                  ;Save entry column
60C9: A9 03                LDA    #3
60CB: 20 5B FB             JSR    TABV
60CE: A9 05                LDA    #5
60D0: 85 24                STA    CH
60D2: AC F6 61             LDY    N             ;Slot offset
60D5: B9 78 05             LDA    XH,Y          ;Hi byte X-coordinate
60D8: BE 78 04             LDX    XL,Y          ;Lo byte X-coordinate
60DB: 20 24 ED             JSR    LINPRT        ;Print X-coordinate
60DE: 20 48 F9             JSR    PRBLNK
60E1: A9 0F                LDA    #15
60E3: 85 24                STA    CH
60E5: AC F6 61             LDY    N             ;Slot offset
60E8: B9 F8 05             LDA    YH,Y          ;Hi byte Y-coordinate
60EB: BE F8 04             LDX    YL,Y          ;Lo byte Y-coordinate
60EE: 20 24 ED             JSR    LINPRT        ;Print Y-coordinate
60F1: 20 48 F9             JSR    PRBLNK
60F4: A9 1A                LDA    #26
60F6: 85 24                STA    CH
60F8: AC F6 61             LDY    N             ;Slot offset
60FB: B9 78 07             LDA    BUTTON,Y
60FE: A2 08                LDX    #8            ;Bit counter
6100: 0A          :8       ASL
6101: 48                   PHA
6102: 90 03                BCC    :9            ;Clear bit found
6104: A9 B1                LDA    #"1"          ;Set bit found
6106: 2C                   HEX    2C            ;Skip next 2 bytes
6107: A9 B0       :9       LDA    #"0"
6109: 20 ED FD             JSR    COUT          ;Print bit status
610C: 68                   PLA
610D: CA                   DEX                  ;Decrement bit counter
610E: 10 F0                BPL    :8            ;Get another bit
6110: 68                   PLA
6111: 85 24                STA    CH            ;Restore entry column
6113: 68                   PLA
6114: 4C 5B FB             JMP    TABV          ;Restore entry row
                  *-------------------------------
                  * Call Mouse Firmware:
                  *-------------------------------
                  * Entry conditions:
                  *     X = Cn
                  *     Y = n0
                  *     A = user defined

6117: 48          CALLFIRM PHA
6118: B1 06                LDA    (PTR),Y       ;Set lo byte of Mouse
611A: 8D 25 61             STA    FIRMADR+1     ; firmware routine
611D: AE F7 61             LDX    CN            ;Entry X-reg
6120: AC F8 61             LDY    N0            ;Entry Y-reg
6123: 68                   PLA                  ;Entry A-reg
6124: 4C 00 00    FIRMADR  JMP    $0000         ;Set by CHKMOUSE & CALLFIRM
                  *-------------------------------
                  * Format Screen:
                  *-------------------------------
6127: 20 58 FC    FMTSCR   JSR    HOME
612A: A2 00                LDX    #0
612C: BD 6B 61    :A       LDA    TXHDR,X       ;Print header
612F: F0 06                BEQ    :B
6131: 20 ED FD             JSR    COUT
6134: E8                   INX
6135: D0 F5                BNE    :A            ;Always
6137: A9 03       :B       LDA    #3
6139: 20 5B FB             JSR    TABV
613C: A9 03                LDA    #3
613E: 85 24                STA    CH
6140: A9 D8                LDA    #"X"          ;Print status line
6142: 20 ED FD             JSR    COUT
6145: A9 BD                LDA    #"="
6147: 20 ED FD             JSR    COUT
614A: A9 0D                LDA    #13
614C: 85 24                STA    CH
614E: A9 D9                LDA    #"Y"
6150: 20 ED FD             JSR    COUT
6153: A9 BD                LDA    #"="
6155: 20 ED FD             JSR    COUT
6158: A9 17                LDA    #23
615A: 85 24                STA    CH
615C: A9 C2                LDA    #"B"
615E: 20 ED FD             JSR    COUT
6161: A9 BD                LDA    #"="
6163: 20 ED FD             JSR    COUT
6166: A9 A5                LDA    #"%"
6168: 4C ED FD             JMP    COUT

616B: AA AA AA    TXHDR    ASC    "***** APPLEMOUSE TRACKING STATION *****"
616E: AA AA A0 C1 D0 D0 CC C5
6176: CD CF D5 D3 C5 A0 D4 D2
617E: C1 C3 CB C9 CE C7 A0 D3
6186: D4 C1 D4 C9 CF CE A0 AA
618E: AA AA AA AA
6192: 00                   DFB    00
                  *-------------------------------
                  * Check Slots for Mouse Firmware:
                  *-------------------------------
                  * Signature bytes of Mouse firmware:
                  *     Cn0C = $20
                  *     CnFB = $D6

                  * Look for Mouse firmware:

6193: A2 08       CHKMOUSE LDX    #8            ;Slot counter (+1)
6195: A9 00                LDA    #0            ;Lo byte of Cn00
6197: 85 06                STA    PTR
6199: A9 C8                LDA    #$C8          ;Hi byte of Cn00 (+1)
619B: 85 07                STA    PTR+1
619D: C6 07       :C       DEC    PTR+1         ;Decrement Cn
619F: CA                   DEX                  ;Decrement slot counter
61A0: F0 23                BEQ    NOMOUSE       ;Mouse firmware not found
61A2: A0 0C                LDY    #$C           ;Offset to Cn0C
61A4: B1 06                LDA    (PTR),Y       ;Get byte
61A6: C9 20                CMP    #$20          ;Is it 1st ID byte?
61A8: D0 F3                BNE    :C            ;No. Check next slot
61AA: A0 FB                LDY    #$FB          ;Offset to CnFB
61AC: B1 06                LDA    (PTR),Y       ;Get byte
61AE: C9 D6                CMP    #$D6          ;Is it 2nd ID byte?
61B0: D0 EB                BNE    :C            ;No. Check next slot

                  * Mouse firmware found:

61B2: A5 07                LDA    PTR+1
61B4: 8D 26 61             STA    FIRMADR+2     ;Set hi byte of slot
61B7: 8D F7 61             STA    CN            ;Save Cn for X-reg
61BA: 0A                   ASL                  ;Shift n to hi nibble
61BB: 0A                   ASL
61BC: 0A                   ASL
61BD: 0A                   ASL
61BE: 8D F8 61             STA    N0            ;Save n0 for Y-reg
61C1: 8E F6 61             STX    N             ;Save slot #
61C4: 60                   RTS

                  * Mouse firmware not located:

61C5: 20 58 FC    NOMOUSE  JSR    HOME
61C8: A2 00                LDX    #0
61CA: BD D8 61    :D       LDA    TXNOMSE,X     ;Print message
61CD: F0 06                BEQ    TOBASIC
61CF: 20 ED FD             JSR    COUT
61D2: E8                   INX
61D3: D0 F5                BNE    :D            ;Always
61D5: 4C D0 03    TOBASIC  JMP    DOSWARM

61D8: 87 8D       TXNOMSE  HEX    878D
61DA: CD CF D5             ASC    "MOUSE FIRMWARE NOT FOUND..."
61DD: D3 C5 A0 C6 C9 D2 CD D7
61E5: C1 D2 C5 A0 CE CF D4 A0
61ED: C6 CF D5 CE C4 AE AE AE
61F5: 00                   DFB    00
                  *-------------------------------
                  * Storage Locations:
                  *-------------------------------
                  N        DS     1,0           ;Slot #
                  CN       DS     1,0           ;X-reg setup
                  N0       DS     1,0           ;Y-reg setup
                  OLDCHAR  DS     1,0           ;Screen char replaced
                                                ;by cursor


--End assembly--

506 bytes

Errors: 0
```

END OF LISTING 2

```
          KEY PERFECT 4.0
               RUN ON
            MOUSE.TRACK
    ===========================
      CODE      ADDR# - ADDR#
      ------    -------------
      2BA2      6000 - 604F
      26CF      6050 - 609F
      2E1A      60A0 - 60EF
      271B      60F0 - 613F
      2C31      6140 - 618F
      29B0      6190 - 61DF
      0E20      61E0 - 61F5
    PROGRAM CHECK IS : 01F6
```