Mouse
#1: Interrupt Environment with the Mouse

Revised by: Matt Deatherage November 1988
Revised by: Rilla Reynolds November 1985

This Technical Note describes the interrupt environment one should take into account when programming mouse-based applications on the Apple II family of computers.

Software developers who are writing mouse-based programs in assembly language need to be concerned about the computer's interrupt environment, even if they are using the mouse in passive mode. Listed below are several conditions which assembly language programmers should take into account if their programs are to run on the Apple II family of computers.

- Do not disable interrupts unless absolutely necessary. If you disable them, be sure to re-enable them.
- Disable interrupts when calling any mouse routine. Always use `PHP` and `SEI` to disable interrupts, then use `PLP` to re-enable them. This method preserves the state of interrupts (enabled or disabled).
- Do not re-enable interrupts (`PLP`) after a call to `ReadMouse` until X and Y data have been removed from the screen holes.
- Disable interrupts (`PHP` and `SEI`) before placing position information in the screen holes (`PosMouse` or `ClampMouse`).
- Enter all mouse routines (except `ServeMouse`) with the X register set to $Cn and Y register set to $n0, where n = the slot number.
- Some programs need to disable interrupts for purposes other than reading the mouse. If interrupts are disabled then re-enabled, the first call to `ReadMouse` could return incorrect values; subsequent calls to `ReadMouse` will return correct values until interrupts are disabled and re-enabled again. Disabling interrupts for mouse calls does not create this problem. If you watch numbers from the mouse while moving it in a direction which would increase values, you would see something similar to: 6, 7, 8, 9, 8, 9, 10. In practice, this momentary "glitch" in the stream of data has little importance. If you feel you must avoid this glitch altogether, do not disable interrupts for more than 40 microseconds or make sure that at least one mouse interrupt takes place after re-enabling interrupts.

Mouse
#2: Varying VBL Interrupt Rate

Revised by: Matt Deatherage November 1988
Revised by: Rilla Reynolds November 1985

This Technical Note describes a method to make the AppleMouse peripheral card interrupt at a rate other than the default 60 Hz. This method does not work on the Apple IIc or IIGS.

This Technical Note describes a previously undocumented call to the AppleMouse II firmware which allows the user to set the interrupt rate to 50 or 60 Hz. (The default is 60 Hz, which keeps the card-generated VBL interrupts synchronized with the actual VBL rate on standard North American Apples; European Apples use 50 Hz as a standard.)

Call:	`TimeData`		
Offset Location:	$Cn1C		
Input:	Accumulator bit 0:	0 for 60 Hz	
		1 for 50 Hz	

Note: All other accumulator bits are reserved, and **must** be set to 0.

Output:	carry bit clear
	screen holes unchanged

You must make this call just prior to calling `InitMouse` to be effective. If you want to change the interrupt rate in the middle of an application, you must call `TimeData` with the appropriate value in the accumulator, then call `InitMouse` (which generates an interrupt). `InitMouse` resets the mouse position, mode, clamps, etc. to their default values. If you fail to call `TimeData`, `InitMouse` will use a default interrupt rate of 60 Hz.

Note: This call exists **only** on the AppleMouse card for the IIe or][+ and should only be used when you know you are working with a IIe or][+. A user may configure a IIGS to 50 Hz by holding down the Option key while rebooting. The standard North American Apple IIc will not generate 50 Hz VBL interrupts.

Mouse
#3: Mode Byte of the SetMouse Routine

Revised by: Matt Deatherage November 1988
Revised by: Rilla Reynolds November 1985

This Technical Note explains the results of turning the mouse on and off through the mode byte of the `SetMouse` routine.

What Turning the Mouse Off Does

In the description of `SetMouse` and the mouse mode, the low-order bit of the mouse mode is said to control mouse off and mouse on. This terminology is somewhat misleading. When this bit is set to 0, the mouse is off only in the following respects:

1. The mouse position is not tracked; any mouse motion is ignored.
2. `ReadMouse` calls do not update the status byte or the screen holes, except on the IIGS, where `ReadMouse` always functions the same, regardless of mouse on or mouse off.
3. Button and movement interrupts are not generated, regardless of the other mouse mode bits. Pure VBL interrupts can still be generated, however, if bit 3 is set.

What Turning the Mouse Off Does Not Do

Other mouse functions will continue to work as usual when the mouse is off. `PosMouse` and `ClearMouse` will change the mouse position, `ClampMouse` will set new clamp values, etc. In particular:

1. Turning the mouse off and on with the mode byte does not reset any mouse values, including position. The mouse position retains the last values it had before the mouse was turned off until it is turned on again.
2. A mode byte of $08 (mouse off but VBL interrupt on) will generate VBL interrupts.

Further Reference

- *Apple IIGS Firmware Reference*
- *Apple IIe Technical Reference Manual*

- *Apple IIc Technical Reference Manual, Second Edition*

Mouse
#4: Mouse Firmware Bug Affecting ServeMouse

Revised by: Matt Deatherage November 1988
Revised by: Rilla Reynolds January 1985

This Technical Note documents a bug in the mouse firmware on the AppleMouse card which affects the way `ServeMouse` works.

There is a bug in the AppleMouse II 6805 firmware which may affect the way `ServeMouse` works in an application. If the application takes more than one video cycle (normally about 16 ms) to respond to a mouse-generated interrupt, then `ServeMouse` will not claim the interrupt. The 6805 returns an interrupt status byte of $00 (i.e., no mouse interrupt pending), and the 6502 firmware sets the carry bit (although the interrupt is also cleared by the `ServeMouse` call). This situation can be confusing, and under ProDOS or Pascal it can be lethal. We have identified the following solutions, any of which should work:

If you are **not** working under an established operating system (i.e., ProDOS or Pascal):

1. Do not allow unclaimed interrupts to be fatal to your application. Ignore them.
2. Always service mouse interrupts within 1/60 of a second. If you are forced to disable interrupts for a longer period, first use `SetMouse` to set the mouse mode to 0, then call `ServeMouse` to clear any existing mouse interrupt. After interrupts are re-enabled, restore the mouse mode.

If you **are** working under an established operating system (i.e., ProDOS or Pascal) for which unclaimed interrupts are fatal and the mouse is **not** the only interrupting device:

1. Write the mouse interrupt handler to claim all unclaimed interrupts and make sure the mouse interrupt handler is installed last, otherwise the interrupt will never get through to any interrupt handlers which follow that of the mouse.

Note: This solution may cause cursor flicker by delaying the application's response to VBL interrupts.

2. Write a spurious interrupt handler (also known as a "daemon"), not associated with any device, which claims all unclaimed interrupts (i.e., clears the carry bit then exits). For the reason just mentioned, this interrupt handler must be installed **last**.

Note: Under ProDOS, this limits the number if interrupting devices to three.

This bug exists in the AppleMouse card, therefore you must deal with it when you are writing eight-bit programs for the Apple][+, IIe, IIc and IIGS which use the mouse. The Apple IIGS does not have this bug in its internal mouse firmware, so sixteen-bit “native” mode programs are not affected by it.

Apple II Technical Notes

Developer Technical Support

Mouse
#5: Check on Mouse Firmware Card

Revised by: Matt Deatherage November 1990
Revised by: Rilla Reynolds November 1985

This Technical Note formerly described a protocol which allowed applications to check a device which matched the mouse firmware identification for support of interrupts.
Changes since November 1988: Added the mouse ID bytes since they are no longer included in other documentation.

The convention formerly described by this Note has been removed since it conflicted with the Pascal 1.1 Firmware Protocol. The conflict could cause Pascal to believe that optional firmware routines were present, when the card being checked was simply stating that it supported interrupts.

Apple recommends that any mouse-type device which matches the mouse ID bytes should support interrupts exactly as the Apple mouse firmware does. Applications which believe they have found an Apple mouse have a reasonable right to expect that the device they actually have found behave as an Apple mouse.

In addition to the standard Pascal 1.1 Firmware Protocol ID bytes, the AppleMouse II card is identified by a value of $20 at $Cn0C (“X-Y Pointing device, type zero”) and a value of $D6 at $CnFB, where n is the slot number. The $CnFB value is **not** part of the Pascal 1.1 Firmware Protocol.

Mouse
#6: MouseText Characters

Revised by: Matt Deatherage — January 1989
Revised by: Rilla Reynolds — November 1985

This Technical Note describes the MouseText character set which is available on all currently produced Apple II computers.
Changed since November 1988: Corrected typographical errors in the BASIC and assembly language program examples.

In unenhanced Apple IIe computers, the alternate character set contained two sets of inverse uppercase characters. In the enhanced Apple IIe, and in all Apple IIc and IIGS computers, one set of inverse uppercase characters is replaced by a MouseText character set. MouseText is a set of graphical characters designed to allow Apple II computers to display a desktop metaphor on the text screen. The Apple II Desktop Toolkit uses these characters, as do applications like AppleLink–Personal Edition.

If your program used the set of inverse uppercase characters which were replaced by MouseText (the set mapped to ASCII values $40-$5F), your program will display MouseText characters instead of inverse uppercase characters on all currently-produced Apple II computers. If your program used the other set of inverse uppercase characters (ASCII values $00-$1F), it will display inverse capital characters as expected.

The following will help you identify if the changes affect you or not.

1. If your program is written entirely in BASIC or Pascal or your assembly language program calls the `COUT` routine to put characters on the screen, you are not affected. The only exception would be if you print (`POKE`) inverse characters directly to the text screen in BASIC.
2. If your program uses the standard character set (checkerboard cursor) you are not affected.
3. If your program is using the alternate character set (solid cursor) and is directly storing values (via `POKE`) to the text display area, you will encounter problems if your character values are in the range from 64 ($40) to 95 ($5F). To recreate the original display, use values in the range from 0 ($0) to 31 ($1F) instead. Note that these lower values display as inverse uppercase characters on older machines as well.

Following are the methods recommended for accessing MouseText characters from various languages:

AppleSoft BASIC

1. Turn on the video firmware with `PR#3` (if under DOS 3.3 or ProDOS, use `PRINT CHR$(4);"PR#3"`)
2. Enable MouseText characters by printing an ASCII 27 ($1B) to the screen.
3. Set inverse printing mode by printing an ASCII 15 ($0F) to the screen.

To stop displaying MouseText characters:

1. Disable MouseText characters by printing an ASCII 24 ($18) to the screen.
2. Set normal print mode (if desired) by printing an ASCII 14 ($0E) to the screen.

This short BASIC program displays all MouseText characters under DOS 3.3 and ProDOS:

```
10      D$=CHR$(4)
20      PRINT D$;"PR#3": REM Turn on the video firmware
30      PRINT:REM This is so the screen won't be in inverse
40      PRINT CHR$(15):REM Set inverse mode
50      PRINT CHR$(27);"ABCEDFGHIJKLMNOPQRSTUVWXYZ@[]^_\";CHR$(24)
60      PRINT CHR$(14):END
```

Assembly Language

Assembly language programs are expected to follow the same procedure as AppleSoft BASIC. Use calls to `COUT` to print MouseText characters to the screen. The following is a sample assembly language program which displays two MouseText characters (which create a folder icon), along with their inverse uppercase equivalents:

```
START                   LDA #$A0            ;USE A BLANK SPACE TO
                        JSR $C300           ;TURN ON THE VIDEO FIRMWARE
                        LDY #0              ;INITIALIZE COUNTER
LOOP                    LDA STR,Y           ;GET VALUE
                        JSR $FDED           ;SEND IT THROUGH THE COUT ROUTINE
                        INY
                        CPY STRLEN
                        BNE LOOP            ;=>NOT DONE YET
                        RTS
STR                     DFB $1B,$58,$59,$18,$58,$59
                                            ;MOUSETEXT ON, SHOW, MOUSETEXT OFF, SHOW
STRLEN                  EQU *-STR           ;LENTGH OF STR
```

Note: Using MouseText on the text screen by directly poking or storing MouseText character values into the text buffer is not supported by Apple at this time. Should the MouseText character set require remapping in the future, those programs which use the methods outlined in this Note should still work with any new mapping. Those which directly store MouseText values run the strong risk of display failure under a new mapping.

Apple II Pascal

1. Output a CHR(27), an escape character, to enable MouseText.
2. Output a CHR(15) to turn on inverse video.
3. Output the appropriate capital letter for the desired MouseText character.

A Pascal sample program:

```
PROGRAM OUTPUT_MOUSETEXT
VAR CMD:PACKED ARRAY[0..1] OF 0..255
BEGIN
       CMD[0]:=27; CMD[1]:=15;
       UNITWRITE(1,CMD,2); {turn on MouseText mode}
       {code to display MouseText
                            .
                            .
                            .
       }
       CMD[0]:=24;
       UNITWRITE(1,CMD,1); {turn off MouseText mode}
END
```

Pictorial descriptions of the MouseText character set may be found in the *Apple IIe Technical Reference Manual*, the *Apple IIc Technical Reference Manual, Second Edition*, and the *Apple IIGS Hardware Reference*.

Note: The pictures of MouseText characters in these manuals differ from early implementations. In early MouseText character sets, the icons mapped to the letters F and G combined to form a “running man.” In current production, these letters are different pictures (an inverse carriage return symbol and a window title bar pattern) which form no picture when placed next to each other. Programs should not attempt to use the running man MouseText characters.

Further Reference

- *Apple IIGS Hardware Reference*
- *Apple IIe Technical Reference Manual*
- *Apple IIc Technical Reference Manual, Second Edition*

Apple II Technical Notes

Developer Technical Support

Mouse
#7: Mouse Clamping

Revised by: Matt Deatherage November 1988
Written by: Rilla Reynolds October 1986

This Technical Note describes the different methods available for obtaining mouse clamping values on different Apple II family machines.

AppleMouse Card

The AppleMouse card delivers clamping values on request. There is no specific mouse routine to obtain the clamping values, but an internal routine may be used by the mouse card to return them. The values are returned as minimum and maximum values of X and Y clamps, both low and high bytes.

Note: The following code is the **only** supported use of the $Cn1A offset into the mouse card firmware, and this entry point is not available in any other mouse firmware implementation.

```
GetClamp      LDA   #$4E
              STA   $478              ;Needed by Mouse Card firmware
              LDA   #$00
              STA   $4F8              ;Needed by Mouse Card firmware
              STA   Tmp               ;Zero-page word for indirect addressing
              LDA   #CN               ;$C<slot>, obtained prior to this rtn
              STA   Tmp+1             ;$C<slot>00, Mouse Card firmware main entry
              STA   ToCard+2
              LDY   #$1A
              LDA   (Tmp),Y
              STA   ToCard+1          ;Mouse Card firmware GetClamp entry
              LDA   #7
              STA   BytePtr
              LDY   #N0               ;$<slot>0, for Mouse Card firmware
GetByte       LDX   #CN               ;$C<slot>, for Mouse Card firmware
              LDA   #0                ;Needed by Mouse Card firmware
              JSR   ToCard
              LDA   $578              ;Clamp byte returned by Mouse Card firmware
              LDX   BytePtr
              STA   Byte,X
              DEC   $478
              DEX
              STX   BytePtr
              BPL   GetByte
              RTS
ToCard        JMP   $0000             ;Operand modified by rtn
```

```
Byte           DS     8,0
               ;MinXH,MinYH,MinXL,MinYL,MaxXH,MaxYH,MaxXL,MaxYL
BytePtr        DS     1,0
```

Apple IIc

For the Apple IIc, you can get clamping values by reading the following auxiliary memory screen holes:

$47D	`MinXL`	$67D	`MaxXL`
$4FD	`MinYL`	$6FD	`MaxYL`
$57D	`MinXH`	$67D	`MaxXH`
$5FD	`MinYH`	$6FD	`MaxYH`

Apple IIGS

On the Apple IIGS, the Miscellaneous Tool Set call `GetMouseClamp` returns the mouse clamp values as four words on the stack. This call is documented in the *Apple IIGS Toolbox Reference*, Volume 1.

Further Reference

- *Apple IIGS Toolbox Reference*, Volume 1