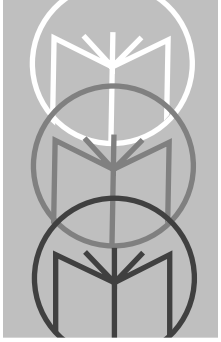

Chapter 5 Mouse Emulator

Introduction

Mouse emulator software has been developed for use on PPT 41XX terminals as a Terminate and Stay Resident (TSR) program to support pen-aware applications. The mouse emulator software (PEN4100.EXE) along with associated files is shipped with the PPT 41XX Software Development Kit (SDK) and must be loaded on the terminal prior to using the terminal to run DOS mouse-aware applications using the pen.

The main topic of this chapter is the use of the mouse emulator API to process function calls through **INT 0x33** from pen-aware application programs. This is the same software interrupt as that used by the Microsoft Mouse driver.

No special action is required to activate mouse emulation on the PPT 41XX. When a pen-aware application is loaded on the terminal, the pen functions the way a mouse usually does. For example, the user issues mouse points by touching points on the screen with the pen to move the cursor or touching menus and choice boxes with the pen to select items for display, execution, etc. The tip of the pen acts as the left mouse button. In *pen down* state (i.e., when the pen touches the tablet), the mouse emulator performs the analogous function(s) associated with pressing the left mouse button. The mouse emulator interprets the *pen up* state (i.e., when the pen is *not* touching the terminal screen) as analogous to the release of the left mouse button. Unlike a mouse, however, the “mouse position” reported never changes unless the pen is down.

In effect, the mouse emulator TSR program provided with the PPT 41XX replaces Microsoft Mouse functions that support mouse-aware applications with its own compatible API. Refer to the **Supported API Commands** section, below, for a list and descriptions of these mouse emulator functions.

Usually, the mouse emulator is loaded as a TSR from the AUTOEXEC.BAT file when the terminal is booted. If the appropriate statements have not been placed in the AUTOEXEC.BAT file or if you choose not to use this method of invoking the mouse emulator, you can change to the directory in which PEN4100.EXE file has been stored on the terminal and enter:

```
PEN4100
```

on the command line. This loads the mouse emulator as a TSR program and makes available to application programs the API described in *Supported API Commands*.

Supported API Commands

PEN4100.EXE is the TSR program that provides the API for applications that need to emulate mouse services on the PPT 41XX terminal. It provides access to a set of pen functions which pass to the application information such as whether the pen is up or down, its current position, etc. All services provided by the mouse emulator are accessed through software interrupt 0x33.

This section consists of the following subsections:

- *Mouse Emulator API Commands (Listing)* lists the API commands supported by the mouse emulator.
- *Mouse Emulator API Commands (Descriptions)* provides individual descriptions of the API commands (functions) supported by the mouse emulator.

Mouse Emulator API Commands (List)

Table 5-1 lists the functions supported by the mouse emulator TSR program on the PPT 41XX terminal. The list is sorted by the hexadecimal numeral for the function code that an application must assign to the AX register prior to calling **INT 0x33** to request the desired service. Descriptions of the functions in Table 5-1 are provided in the *Command Descriptions* subsection that follows the table.

Note: Issuing a Mouse Emulator API command other than one of those listed in Table 5-1 returns with register contents unchanged.

Table 5-1. Mouse Emulator API Commands (INT 0x33)

Function Code	Mouse Emulator Service Name
0x00	*Reset Mouse Emulator*
0x03	*Get Pen State and Pen Position*
0x05	*Get Pen Down Information*
0x06	*Get Pen Up Information*
0x07	*Set Horizontal Pen Limits*
0x08	*Set Vertical Pen Limits*
0x0B	*Read Pen Motion Counters*
0x0C	*Set User-Defined Pen Event Handler*
0x14	*Swap User-Defined Pen Event Handlers*
0x24	*Get Mouse Emulator Information (Version Number, Type, and IRQ Number)*)

Mouse Emulator API Commands (Descriptions)

The following descriptions of the Mouse Emulator API commands (functions) in Table 5-1 are given in function code order.

Reset Mouse Emulator

Function: 0x00

Description

Resets or initializes the mouse emulator and returns its status.

Use this function to initialize the mouse (pen) and prepare it for its first use.

This function returns (in the AX register) the status of the mouse emulator (i.e., installed / not installed) and sets defaults.

The value returned for the number of buttons (in the BX register) should be one (0x0001); that is, the pen down position emulates a single button mouse on PPT 41XX terminals on which the mouse emulator has been installed.

Interrupt

0x33

Input Registers

AX = 0x00

Output Registers

AX = Status of mouse emulator as follows:

0x0000: Hardware / Driver (Mouse Emulator) not installed
0xFFFF: Hardware / Driver (Mouse Emulator) installed

BX = 0x0001

Note: The value returned in BX indicates the number of mouse buttons supported by the installed mouse driver. In this case, this value should be one, since the mouse emulator supports only the pen down position as emulating a mouse button.

Notes

Mouse-aware applications should issue a **reset** before using the mouse, particularly important with PEN4100.EXE which may not work properly in all cases unless the **reset** call is made. If a recalibration operation is performed, the **reset** call *must be used* to restore proper operation.

Example

For a code sample that illustrates the use of the **Reset Mouse Emulator** service, refer to the *Appendix* at the end of this chapter.

Get Pen State and Pen Position

Function: 0x03

Description

Returns current pen state (up / down) and pen position (in virtual screen coordinates). See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x03

Output Registers

BX = Pen state as follows:

0x00: Pen is up
0x01: Pen is down

CX = Horizontal pen position (See **Notes** section.)

DX = Vertical pen position (See **Notes** section.)

Notes

The coordinates for the pen positions are based on virtual screen size. The virtual screen size is initially set by the video mode call as indicated in the following chart:

Video Mode	Virtual Screen Size
02	640 x 200
03	640 x 200

An application can modify virtual screen size by calling functions **0x07** (**Set Horizontal Pen Limits**) and **0x08** (**Set Vertical Pen Limits**).

Example

For a code sample that illustrates the **Get Pen State and Position** service, refer to the *Appendix* at the end of this chapter.

Get Pen Down Information

Function: 0x05

Description

Returns pen state (up or down) and pen position (in virtual screen coordinates). See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x05

BX = 0x00

Output Registers

AX = Pen state as follows:

0x00: Pen is up
0x01: Pen is down

BX = Number of Pen Down's since last call of **Function 0x05**

CX = Horizontal pen position at Pen Down (See **Notes** section, below.)

DX = Vertical pen position at Pen Down (See **Notes** section, below.)

Notes

The coordinates for the pen positions are based on virtual screen size. The virtual screen size is initially set by the video mode call as indicated in the following chart:

Video Mode	Virtual Screen Size
02	640 x 200
03	640 x 200

An application can modify virtual screen size by calling functions **0x07** (**Set Horizontal Pen Limits**) and **0x08** (**Set Vertical Pen Limits**).

Example

For a code sample that illustrates the **Get Pen Down Information** service, refer to the *Appendix* at the end of this chapter.

Get Pen Up Information

Function: 0x06

Description

Returns pen state (up or down) and pen position (in virtual screen coordinates). See **Notes** section, below.

Interrupt

0x33

Input Registers

AX = 0x06

Output Registers

AX = Pen state as follows:

0x00: Pen is up
0x01: Pen is down

BX = Number of Pen Up's since last call of **Function 0x06**

CX = Horizontal pen position at last Pen Up (See **Notes** section.)

DX = Vertical pen position at last Pen Up (See **Notes** section.)

Notes

The coordinates for the pen positions are based on virtual screen size. The virtual screen size is initially set by the video mode call as indicated in the following chart:

Video Mode	Virtual Screen Size
02	640 x 200
03	640 x 200

An application can modify virtual screen size by calling functions **0x07** (**Set Horizontal Pen Limits**) and **08h** (**Set Vertical Pen Limits**).

Example

For a code sample that illustrates the **Get Pen Up Information** service, refer to the *Appendix* at the end of this chapter.

Set Horizontal Pen Limits

Function: 0x07

Description

Restricts the horizontal movement of the pen position to a specified area on the screen.

The position values specified in CX and DX should be within the valid range of horizontal values for the current screen mode. See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x07

CX = Minimum horizontal position (See **Notes** section, below.)

DX = Maximum horizontal position (See **Notes** section, below.)

Output Registers

None

Notes

The coordinates for the pen positions are based on virtual screen size. The virtual screen size is initially set by the video mode call as indicated in the following chart:

Video Mode	Virtual Screen Size
02	640 x 200
03	640 x 200

Example

For a code sample that illustrates the **Set Horizontal Pen Limits service**, refer to the *Appendix* at the end of this chapter.

Set Vertical Pen Limits

Function: 0x08

Description

Restricts the vertical movement of the pen position to a specified area on the screen.

The position values specified in CX and DX should be within the valid range of vertical values for the current screen mode. See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x08

CX = Minimum vertical position (See **Notes** section.)

DX = Maximum vertical position (See **Notes** section.)

Output Registers

None

Notes

The coordinates for the pen positions are based on virtual screen size. The virtual screen size is initially set by the video mode call as indicated in the following chart:

Video Mode	Virtual Screen Size
02	640 x 200
03	640 x 200

Example

For a code sample that illustrates the **Set Vertical Pen Limits** service, refer to the *Appendix* at the end of this chapter.

Read Pen Motion Counters

Function: 0x0B

Description

Returns the relative change of the pen position since the last time this function was called. See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x0B

Output Registers

CX = Horizontal relative motion count in mickeys

DX = Vertical relative motion count in mickeys

Notes

It is strongly recommended that this function *not* be used in pen-based applications, since the concept of relative motion makes little sense with an absolute pointing device like the pen. It has been implemented primarily in an attempt to be compatible with the existing base of applications that do use it.

Example

For a code sample that illustrates the **Read Pen Motion Counters** service, refer to the *Appendix* at the end of this chapter.

Set User-Defined Pen Event Handler

Function: 0x0C

Description

Enables an application to tack on its own subroutine after certain pen actions.

Bit settings in the mask passed in the CX register specify the pen action on which the subroutine should be called. The value passed in the ES:DX register pair contains the address of the far routine to be called when the specified pen action occurs. See **Notes** section.

Interrupt

0x33

Input Registers

AX = 0x0C

CX = Call mask with bit settings as follows:

Bit 0 = Pen position changed
Bit 1 = Pen down event
Bit 2 = Pen up event
Bits 3 through 15 are not used.

ES:DX = Segment and offset of the subroutine address

Output Registers

None

Notes

When the mouse emulator calls the subroutine the application has identified in this function call, it loads the following information into the processor's registers:

AX = Condition mask (similar to the call mask input in CX by the application except that a bit is set only if the condition associated with it in the call mask occurs)

BX = Pen state as follows:

0x00: pen is up
0x01: pen is down

CX = Horizontal pen position

DX = Vertical pen position

SI = Horizontal relative motion count in mickeys

DI = Vertical relative motion count in mickeys

DS is not used.

Example

For a code sample that illustrates the **Set User-Defined Pen Event Handler** service, refer to the *Appendix* at the end of this chapter.

Swap User-Defined Pen Event Handlers

Function: 0x14

Description

Replaces an address and a mask for an interrupt subroutine set by a previous call to **Function 0x0C** (see **Set User-Defined Pen Event Handler**) or to this function with the address and mask for a different interrupt subroutine required in an application program.

Bit settings in the mask passed in the CX register specify the pen action on which the subroutine should be called. The value passed in the ES:DX register pair contains the address of the far routine to be invoked when the pen action occurs.

Upon return, the address of the old interrupt subroutine and the interrupt mask that invoked it are in the ES:DX and CX registers and can be stored for future use.

Interrupt

0x33

Input Registers

AX = 0x14

CX = Call mask with bit settings as follows:

Bit 0 = Pen position changed
Bit 1 = Pen down event
Bit 2 = Pen up event
Bits 3 through 15 are not used.

ES:DX = Segment and offset of the subroutine address

Output Registers

CX = Call mask of the previous interrupt routine

ES:DX = Far address of the previous interrupt routine

Notes

When the mouse emulator calls the subroutine the application has identified in this function call, it loads the following information into the processor's registers:

AX = Condition mask (similar to the call mask input in CX by the application except that a bit is set only if the condition associated with it in the call mask occurs)

BX = Pen state as follows:

0x00: pen is up
0x01: pen is down

CX = Horizontal pen position

DX = Vertical pen position

SI = Horizontal relative motion count in mickeys

DI = Vertical relative motion count in mickeys

DS is not used.

Notes

It is strongly recommended that this function *not* be used in pen-based applications, since the concept of relative motion makes little sense with an absolute pointing device like the pen. It has been implemented to be compatible with the existing base of applications that do use it.

Example

For a code sample that illustrates the **Swap User-Defined Pen Event Handlers** service, refer to the *Appendix* at the end of this chapter.

Get Mouse Emulator Information (Version Number, Type, and IRQ Number)

Function: 0x24

Description

Returns the mouse emulator version number, the mouse emulation type, and the IRQ number.

Interrupt

0x33

Input Registers

AX = 0x24

Output Registers

BH = 0x06 (Mouse emulator major version number)

BL = 0x26 (Mouse emulator minor version number)

CH = 0x02 (i.e., serial type)

CL = 0x05 (IRQ number)

Example

For a code sample that illustrates the **Get Mouse Emulator Information** service, refer to the *Appendix* at the end of this chapter.

Appendix. PENMOUSE.C

This appendix contains the code samples referred to in **Example** sections of API function descriptions in the **Mouse Emulator API Commands (Descriptions)** section of this chapter. It is also contained in the SDK file

C:\SDK4100\SAMPLES\MANUAL\CHAP5\PENMOUSE.C

where **C:\SDK4100** is the default installation directory.

Note: This application requires that ANSI.SYS be loaded in the CONFIG.SYS file.

```
/* Include Files ************************************************/

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Defines, typdefs, etc. ***************************************/

typedef enum {FALSE, TRUE} boolean;

#define PN_SET_PEN_RESET          0x00    /* Pen/Mouse Driver   */
#define PN_GET_PEN_STATE_INFO     0x03    /* Function codes     */
#define PN_GET_PEN_DOWN_INFO      0x05
#define PN_GET_PEN_UP_INFO        0x06
#define PN_SET_MIN_MAX_HORZ       0x07
#define PN_SET_MIN_MAX_VERT       0x08
#define PN_GET_PEN_POS_REL        0x0B
#define PN_SET_USER_HANDLER       0x0C
#define PN_SWAP_USER_HANDLERS     0x14
#define PN_GET_VERSION_NUMBER     0x24

/*  Define the Services by Interrupt Vector number          */

#define PENMS_INT     0x33          /* Pen   interrupt number */
#define UNDEFINED     0xff          /* Used to check version # */

enum {PEN_UP, PEN_DOWN} pen_state;    /* Used for button status */
```

```
enum {DISABLE_CALLBACK,
      PEN_POSITION_CHANGED,          /* Event mask used by Call- */
      PEN_DOWN_EVENT,                /*  back routine.        */
      PEN_UP_EVENT} pen_event_mask;

/*  ANSI escape sequences                          */

#define CLEAR_SCREEN     "\x1b[2J"      /* clear entire screen      */
#define REVERSE_VIDEO    "\x1b[7m"      /* put text in reverse video */
#define NORMAL_VIDEO     "\x1b[m"       /* put text in normal video */
#define SET_CURSOR       "\x1b["        /* set cursor to row:col    */

/* Public Variables ********************************************/

/* global flag that get sets to TRUE when callback routine       */
/* conditions are met.                                           */

int callbackOccurred = FALSE;      /* callback occurred flag   */

union REGS inregs;                 /* input regs to int86x    */
union REGS outregs;                /* output regs from int86x */
struct SREGS segregs;              /* seg regs to/from int86x */

/* Local Function Prototypes ***********************************/

/*********************************************************************
* SYNOPSIS:    void callbackRoutine(unsigned wParam)
* DESCRIPTION: This routine is called from Pen_TSRCB which
*                    is called from the TSR (Careful about I/O in
*                    this routine).
*                    Stack Checking MUST Be Disabled For This Routine
* PARAMETERS:  CPU register set
* RETURN VALUE: None
* INPUTS: None
* OUTPUTS: None
* NOTES:
*         On entry:       AX = ISR Condition mask
```

```
*                       BX = Button Status
*                       CX = x location
*                       DX = y location
*                       SI = horizontal mickey count
*                       DI = vertical mickey count
* PSEUDOCODE:   Notify the compiler to suppress generating stack
*                       checking by declaring the routine type "interrupt"
*                       Set global flag (callbackOccurred = TRUE) that
*                       call back occurred when conditions are met
*                       FARRETURN to caller, never executing IRET generated
*                        by the compiler.
************************************************************************ /

#ifdef _MSC_VER
#define FAR_RETURN __asm mov sp,bp __asm pop es __asm pop ds __asm popa __asm retf

#pragma optimize ( "g2",on )
void interrupt far callback (int es_Reg, int ds_Reg,
                             int di_Reg, int si_Reg,
                             int bp_Reg, int sp_Reg,
                             int bx_Reg, int dx_Reg,
                             int cx_Reg, int ax_Reg,
                             int ip_Reg, int cs_Reg)
#else
#define FAR_RETURN asm mov sp,bp; asm add sp,12h; asm retf
void interrupt far callback (int bp_Reg,
                             int di_Reg, int si_Reg,
                             int ds_Reg, int es_Reg,
                             int dx_Reg, int cx_Reg,
                             int bx_Reg, int ax_Reg,
                             int ip_Reg, int cs_Reg)

#endif
```

```
{
   if (callbackOccurred == FALSE)
   {
        if (bx_Reg == PEN_DOWN)
        {
             if (cx_Reg < 40 && dx_Reg < 10)
             {
                  callbackOccurred = TRUE;
             }
        }
   }
   FAR_RETURN
}

/********** Reset Mouse Emulator function ********************/

void pn_SetPenReset(signed int far *pen_reset_state)
{
     inregs.x.ax = PN_SET_PEN_RESET;
     int86(PENMS_INT, &inregs, &outregs);
     *pen_reset_state = outregs.x.ax;

}

/********** Get Pen State and Position function ********************/

void pn_GetPenStateandPosition(unsigned char far *pen_state,
                               unsigned int far *horz_pos,
                               unsigned int far *vert_pos)
{
     inregs.x.ax = PN_GET_PEN_STATE_INFO;
     int86(PENMS_INT, &inregs, &outregs);
     *pen_state = outregs.h.bl;
     *horz_pos = outregs.x.cx;
     *vert_pos = outregs.x.dx;
}
```

```
/********** Get Pen Down Information function **********************/

void pn_GetPenDownInformation(unsigned char far *pen_state,
                              unsigned int far *horz_pos,
                              unsigned int far *vert_pos)
{
    inregs.x.ax = PN_GET_PEN_DOWN_INFO;
    inregs.x.bx = 0;
    int86(PENMS_INT, &inregs, &outregs);
    *pen_state = outregs.h.al;
    *horz_pos = outregs.x.cx;
    *vert_pos = outregs.x.dx;
}

/********** Get Pen Up Information function **********************/

void pn_GetPenUpInformation(unsigned char far *pen_state,
                            unsigned int far *horz_pos,
                            unsigned int far *vert_pos)
{
    inregs.x.ax = PN_GET_PEN_UP_INFO;
    inregs.x.bx = 0;
    int86(PENMS_INT, &inregs, &outregs);
    *pen_state = outregs.h.al;
    *horz_pos = outregs.x.cx;
    *vert_pos = outregs.x.dx;
}

/********** Set Horizontal Pen Limits function ******************/

void pn_SetMinandMaxHorizontalPenPosition(unsigned int min_horz_pos,
                                          unsigned int max_horz_pos)
{
    inregs.x.ax = PN_SET_MIN_MAX_HORZ;
    inregs.x.cx = min_horz_pos;
    inregs.x.dx = max_horz_pos;
    int86(PENMS_INT, &inregs, &outregs);
}
```

```
/********** Set Vertical Pen Limits function ******************/

void pn_SetMinandMaxVerticalPenPosition(unsigned int min_vert_pos,
                                        unsigned int max_vert_pos)
{
    inregs.x.ax = PN_SET_MIN_MAX_VERT;
    inregs.x.cx = min_vert_pos;
    inregs.x.dx = max_vert_pos;
    int86(PENMS_INT, &inregs, &outregs);
}

/************ Read Pen Motion Counters function *****************/

void pn_SetMinandMaxVerticalPenPosition(unsigned int min_vert_pos,
                                        unsigned int max_vert_pos)
{
    inregs.x.ax = PN_SET_MIN_MAX_VERT;
    inregs.x.cx = min_vert_pos;
    inregs.x.dx = max_vert_pos;
    int86(PENMS_INT, &inregs, &outregs);
}

/********* Set User-Defined Pen Event Handler function **************/

void pn_SetUserDefinedPenEventHandler(int pen_event_mask,
                                      void far *callback)
{
    inregs.x.ax = PN_SET_USER_HANDLER;
    inregs.x.cx = pen_event_mask;
    inregs.x.dx = FP_OFF(callback);
    segregs.es = FP_SEG(callback);
    int86x(PENMS_INT, &inregs, &outregs, &segregs);
}
```

```
/********** Swap User-Defined Pen Event Handlers function ***********/

void pn_SwapUserDefinedPenEventHandlers(int pen_event_mask,
                                        void far *newcallback)
{
    inregs.x.ax = PN_SWAP_USER_HANDLERS;
    inregs.x.cx = pen_event_mask;
    inregs.x.dx = FP_OFF(newcallback);
    segregs.es = FP_SEG(newcallback);
    int86x(PENMS_INT, &inregs, &outregs, &segregs);
}

/******* Get Mouse Emulator Information (Version Number) ********/

void pn_GetVersionNumber (unsigned char far *major,
                          unsigned char far *minor,
                          unsigned char far *serial_t,
                          unsigned char far *irq_n)
{
    inregs.x.ax = PN_GET_VERSION_NUMBER;
    inregs.x.bx = 0xffff;
    int86(PENMS_INT, &inregs, &outregs);
    *major = outregs.h.bh;
    *minor = outregs.h.bl;
    *serial_t = outregs.h.ch;
    *irq_n = outregs.h.cl;
}
```

```
void main()
{

/* Local Variables ************************************************/

    unsigned char major;            /* major version # of PENMS */
    unsigned char minor;            /* minor version # of PENMS */
    unsigned char serial_type;      /* serial type of PENMS     */
    unsigned char irq_num;          /* int request number       */

    int pen_reset_state;

    unsigned char pen_state;        /* pen state info at call   */
    unsigned int horz_pos;
    unsigned int vert_pos;

    unsigned int horz_rel;          /* rel motion counter for x */
    unsigned int vert_rel;          /* rel motion counter for y */
    unsigned char pen_state_dn;     /* pen down state info      */
    unsigned int horz_pos_dn;
    unsigned int vert_pos_dn;

    unsigned char pen_state_up;     /* pen up state info        */
    unsigned int horz_pos_up;
    unsigned int vert_pos_up;

    int min_horz_pos;               /* min-max horizontal limits*/
    int max_horz_pos;

    int min_vert_pos;               /* min-max vertical limits  */
    int max_vert_pos;

    char inbuf[132], *iptr;         /* keyboard input buffer    */

/*****************************************************************/
```

```
/* Check if PENMS is loaded before requesting services        */

if (_dos_getvect(PENMS_INT) == NULL)
{
     fprintf (stderr,"PENMS not loaded.\nProgram aborted.");
     exit(0);
}

/*       Extended call to get PENMS version number          */

pn_GetVersionNumber(&major, &minor, &serial_type, &irq_num);

if (major == UNDEFINED)
{
     fprintf (stderr,"PENMS handler not recognized."
     "\nProgram terminated.");
     exit(0);
}
pn_SetPenReset(&pen_reset_state);
if (pen_reset_state != -1)
{
     fprintf(stderr,"\nPen is not attached\nProgram terminating");
     exit(0);
}


/* clear screen, print info returned from GetVersion call       */

printf(CLEAR_SCREEN);

pen_event_mask = PEN_DOWN_EVENT;
pn_SetUserDefinedPenEventHandler(pen_event_mask, callback);

printf(SET_CURSOR "1;1H" REVERSE_VIDEO "PENMS" NORMAL_VIDEO
     " reported version # as : %x.%02x", major, minor);
printf("\nSerial type as : %d", serial_type);
printf(" and IRQ # as : %d", irq_num);

printf(SET_CURSOR "3;1HEnter Min, Max Horizontal [0,319]"
     "\n<CR> for defaults ");
```

```
/* Get input from keyboard                              */

gets(inbuf);

/* If user entered Horizontal min-max data, try to process it ... */

if (strlen(inbuf) > 2)
{
      min_horz_pos = atoi(inbuf);
      iptr = strchr(inbuf, ',');
      if (iptr != NULL)
           max_horz_pos = atoi(iptr+1);

      /* call the pen driver to set new horizontal defaults.          */

      pn_SetMinandMaxHorizontalPenPosition(min_horz_pos, max_horz_pos);
}

printf(SET_CURSOR "4;1H                                   ");
printf(SET_CURSOR "3;1HEnter Min, Max Vertical [0,479]  "
      "\n<CR> for defaults ");

/* Get input from keyboard                              */

gets(inbuf);

/* If user entered Vertical min-max data, try to process it ...  */

if (strlen(inbuf) > 2)
{
      min_vert_pos = atoi(inbuf);
      iptr = strchr(inbuf, ',');
      if (iptr != NULL)
           max_vert_pos = atoi(iptr+1);

      /* call the pen driver to set new vertical defaults.          */

      pn_SetMinandMaxVerticalPenPosition(min_vert_pos, max_vert_pos);

}
```

```
printf(SET_CURSOR "3;1HTouch pen anywhere on screen,     "
    "\nto exit touch " REVERSE_VIDEO "PENMS" NORMAL_VIDEO " on "
    "top line.");
do
{
    /*              Wait for pen touch                    */
    do
    {
        pn_GetPenStateandPosition(&pen_state, &horz_pos, &vert_pos);
    } while(pen_state == PEN_UP);

    printf(SET_CURSOR "3;1HPen down @ x=%3d, y=%3d              ",
                                    horz_pos, vert_pos);

    pn_ReadPenMotionCounters((unsigned int far *)&horz_rel,
                                    (unsigned int far *)&vert_rel);
    printf("\nRelative motion counters: %4d, %4d",
        horz_rel, vert_rel);

    /* Pen is down, wait for pen to be lifted before proceeding   */
    /* This will prevent the same touch from being processed      */
    /* twice.                                    */

    do
    {
        pn_GetPenStateandPosition(&pen_state, &horz_pos, &vert_pos);
    } while (pen_state == PEN_DOWN);

        pn_GetPenDownInformation(&pen_state_dn, &horz_pos_dn,
                                        &vert_pos_dn);
        printf("\nPen down @ x=%3d, y=%3d", horz_pos_dn, vert_pos_dn);

        pn_GetPenUpInformation(&pen_state_up, &horz_pos_up,
                                    &vert_pos_up);
        printf("\nPen up   @ x=%3d, y=%3d", horz_pos_up, vert_pos_up);

} while(callbackOccurred == FALSE);
```

```
        pen_event_mask = DISABLE_CALLBACK;
        pn_SetUserDefinedPenEventHandler(pen_event_mask, callback);

        printf(SET_CURSOR "1;1H" NORMAL_VIDEO "PENMS"
                " reported version # as : %x.%02x", major, minor);
        printf("\nSerial type as : %3d", serial_type);
        printf(" and IRQ # as : %3d", irq_num);
        printf(SET_CURSOR "6;1H");
    }
```