The MouseTrap Librar

High & Low Level Mouse Contro
Functions for 'C' Program
Version 1.

By James M. Curra
By James M. Curran.
24 Greendale Roa
Cedar Grove, N
07009-131

Table of Contents

Introduction

The MouseTrap library is a collection of functions to control a mouse, designed to be called from a 'C' program. They provide easy access to the low-level functions of the mouse interrupt, as well as a simplified system for high-level control over the mouse. The basic functions are mostly self explanatory and are described in chapter 2 of the document. The high-level functions are a bit more complicated. They are described in Chapter 3 with a tutorial in Chapter 4.

Registration

The MouseTrap Library is copyrighted by James M. Curran. You are granted a limited license to use MouseTrap, fo noncommercial programs. You may, and are in fact encouraged, to copy and distribute it, provided that the following conditions are met: (a) No fee may be charged for copying or distributing, and (b) only the library files (*.LIB) and accompanying documentation are distributed, and only in their original, unmodified form.

Sending a voluntary contribution of $15.00 will appease your guilt, and earn you my undying gratitude. It will also get you a copy of the source code, the Compact (CMOUSE.LIB) & Medium (MMOUSE.LIB) memory model libraries, the missing chapter from this booklet, and other assorted related files. Microsoft C 5.1 & MASM 5.1 are needed to recompile the source files, (unless modified by the user).

Contributions, (and requests for information on commercial licenses) should be sent to:

James M. Curran
24 Greendale Road
Cedar Grove, NJ 07009-1313

Finally, there's only one thing you can say for sure about a "Version 1.0" release of software--- That it will soon be followed by a "Version 1.01" Bug-Fix release. So, all registered user will be sent that version when it's ready. (That's merely cautionary; there are no known bugs at this time).

Warranty

Warranty ? We make no promises that the MouseTrap library will do anything useful for you. Nor do we promise that it WON'T do anything harmful. (Life's Tough; "Want do you want for nothing ? Rubber Biscuit ?")

Chapter 2:

Basic Mouse Control Functions

The eleven primitives that make up the low-level support functions are almost direct calls to the mouse driver interrupt, and are written in 8086 assembler. They were originally derived from a set of 'C' functions given in an article in "The 'C' Gazette" (see references at end), but since then numerous revisions have transformed them. The only thing left are th names of two functions.

Check_Mouse()
Show_Mouse()
Hide_Mouse()
Get_Mouse_Position()
Set_Mouse_Position()
Set_Mouse_Limits()
Get_Mouse_Press()
Get_Mouse_Release()
Set_Mouse_Text_Cursor()

Check_Mouse - Check for the existence and type of Mouse.

Syntax:

```
     #include <moustrap.h>

     mouse_t Check_Mouse(void);
```

Description:

This function initialized the mouse interrupt driver, and must be the first low-level function called (but see Define_Mouse_System()). It checks to see if a mouse is attache (or to be exact, if a mouse device driver is loaded into memory), and if one is present, determines how many buttons it has. This information is return by the function, and is also stored in the global variable "_mouse_there".

Returns Value:

If no mouse is detected, the Check_Mouse function returns 0. If a mouse IS detected, the number of buttons on it is returned. These values are also stored in the global _mouse_there. This can also be used as a TRUE/FALSE indicator.

See Also:

Define_Mouse_System, _mouse_there

Example:

```
  #include <moustrap.h>
  #include <stdio.h>
  main()
  {
    Check_Mouse();
    if (_mouse_there)
        printf("A %d-button Mouse was detected.\n",_mouse_there);
     else
        printf("No mouse was found.\n");
   }
```

Show_Mouse - Display Mouse Cursor.
Hide_Mouse - Hide Mouse Cursor.

Syntax:

```
     #include <moustrap.h>

     void Show_Mouse(void);
     void Hide_Mouse(void);
```

Description:

Show_Mouse causes the mouse cursor to be displayed on the screen. Hide_Mouse cause the mouse cursor to disappear. Neither will have any effect if there is no mouse or Check_Mouse has not been executed yet.

Returns Value:

There is no return value.

See Also:

Check_Mouse, _mouse_there

Example:

```
  #include <moustrap.h>
  #include <stdio.h>
  main()
  {
    Check_Mouse();

     Show_Mouse();
     printf("Look, Ma!  A mouse !");
     getch();       /* Mouse visible until a key is pressed */
     Hide_Mouse();
   }
```

Get_Mouse_Position

Syntax:

```
#include <moustrap.h>
mouse_t Get_Mouse_Position(mouse_t *X, mouse_t *Y);
```

Description:

Get_Mouse_Position places the X (Horizontal) and Y (Vertical) coordinates of the present location of the mouse cursor into the locations given by X & Y. The locations are given using graphic coordinates in the range (0,0) to (639,199). It also returns the binary sum of the buttons pressed. If _mouse_there indicates that no mouse was detected, the values of X & Y are left unchanged, and the function returns 0.

Return Value:

The binary sum of the buttons pressed, where the Left button equals 1, the Right button equals 2, and the Middle button, 4. These values are added together if more than one button is pressed. For example, pressing the Left & Middle buttons would have Get_Mouse_Position return a value of 5.

See Also:

_mouse_there, Set_Mouse_Position

Example:

```
#include <moustrap.h>
#include <stdio.h>
main()
{
    int     X,Y,m;

    Check_Mouse();
    do {
           m = Get_Mouse_Position( &X, &Y);
           if (m & 1)
                  printf("Left Button, ");

           if (m & 2)
                  printf("Right Button, ");

           if (m & 4)
                  printf("Middle Button, ");
           if (m)
                  printf("pressed at (%d, %d)\n",X,Y);
```

```
        } while (m==0);
}
```

Pressing the Left & Right buttons would print something similar to:

```
Left Button, Right Button pressed at (120, 85)
```

Set_Mouse_Position

Syntax:

```
#include <moustrap.h>

void Set_Mouse_Position(mouse_t X, mouse_t Y);
```

Description:

The function Set_Mouse_Position moves the mouse cursor to the screen location given by the graphic coordinates (X,Y). X must be in the range (0-639) and Y in the range (0-199).

Return Value:

None.

See Also:

Get_Mouse_Position

Example:

```
#include <moustrap.h>
#include <stdio.h>

 main()
 {
     int     X,Y,m;

     Check_Mouse();

     Show_Mouse();
     Get_Mouse_Position( &X, &Y);

     X++;
     Y--;

     Set_Mouse_Position(  X,  Y);

 }
```

The above program would move the mouse cursor, "Up" and to the "Right", without the mouse physically being moved.

Get_Mouse_Press
Get_Mouse_Release

Syntax:

```
mouse_t Get_Mouse_Press(mouse_t Button, mouse_t Status,
                               mouse_t *X,     mouse_t *Y);

mouse_t Get_Mouse_Release(mouse_t Button, mouse_t Status,
                               mouse_t *X,     mouse_t *Y);
```

Description:

Get_Mouse_Press returns information about the last press of one of the mouse buttons, given by the code "Button". The coordinates of the location of the mouse cursor the last time that button was pressed are returned in X & Y. The Function itself returns the number of times that button was pressed since the last time Get_Mouse_Press was called. The binary sum of the buttons currently pressed, as described in Get_Mouse_Position, is returned in Status.

Get_Mouse_Release works exactly the same way, with X and Y giving the location of the last position the button was released.

Return Value:

The number of time Button was pressed (released) since the last call to Get_Mouse_Press (Get_Mouse_Release).

See Also:

Get_Mouse_Position

Example:

```
#include <moustrap.h>
#include <stdio.h>

main()
{
     mouse_t s,x,y;
     if (Check_Mouse()) {
          getch();         /* pause of while */
          if (Get_Mouse_Press(M_Left,&s,&x,&y))
          printf("Left buttom was pressed at %d,%d\n", x,y);
          }
}
```

Set_Mouse_Limit	(v1.0)
Set_Mouse_Limit_Horiz	(v1.1)
Set_Mouse_Limit_Vert	(v1.1)
Set_Mouse_Region	(v1.1)

Syntax:

```
void    Set_Mouse_Limits(Direction, Min, Max);
mouse_t Direction;       /* M_HORIZ -or- M_VERT */
mouse_t Min;
mouse_t Max;

void    Set_Mouse_Limits_Horiz(Left, Right);
void    Set_Mouse_Limits_Vert(Top, Bottom);
void    Set_Mouse_Region(Top, Left, Bottom, Right);

mouse_t Top;
mouse_t Left;
mouse_t Bottom;
mouse_t Right;
```

Description:

These functions give a variety of ways to forces the mouse cursor's movements to remain within specified limits. The various edges are given using graphic coordinates (0-629) (0-199).

"That seems like a fairly simple concept", you should now be saying, "So, why does it take -FOUR- separate functions ?" Funny you should ask. In release 1.0, only Set_Mouse_Limit existed. It was functional, but inelegant, since it required two calls to properly limit the cursor. It was written that way mainly because the function it was adapted from was written that way. Realizing that this was a particularly poor reason to do something badly, Set_Mouse_Region was added. It accomplishs the task with just one call, and maintains the format of the other function in the library. However, it's not as flexible as the original, so Set_Mouse_Limits_Horiz, and Set_Mouse_Limits_Vert were added. Set_Mouse_Limits hung around to maintain upward compatiblity with release 1.0

Return Value:

None

Example:

```
     #include <moustrap.h>
     #include <stdio.h>

     main()
     {
          Check_Mouse();

          Set_Mouse_Region(50,160,150,480);
           /* Mouse is now limited to the center of the screen */

          Read_Mouse();

     }
```

Set_Mouse_Text_Cursor

Syntax:

```
void    Set_Mouse_Text_Cursor(Type, P1, P2);

mouse_t Type; /* 1 = Hardware Cursor | 0 = Software Cursor *
mouse_t P1    /* Start scan line     |     Screen Mask     *
mouse_t P2;   /* Stop  scan line     |     Cursor Mask     *
```

Description:

Set_Mouse_Text_Cursor describes how the mouse cursor will appear on the screen while in text modes. This can be done in either of two way: by using the Hardware cursor, or the Software cursor. The Hardware cursor is the same one that the keyboard uses, that is, it looks just like the keyboard cursor, and can be moved using the BIOS "move cursor" functions. If the hardware cursor is used, P1 & P2 give the scan lines for that cursor. For normal screen uses that is 6 & 7.

The Software cursor is a little more complex. There, P is the "Screen mask" and P2 is the "Cursor Mask", and they are given in the form of a color attribute and character. When the software cursor is drawn on the screen, the character and color attribute originally at that location is first ANDed with the screen mask, then the result of that is XORed with the cursor mask. If the screen mask is 0, the net effect is that the current value at that location is replaced by the cursor mask. If the screen mask is nonzero, the current value at the screen location WILL affect of character or color of the mouse cursor.

For example, if the screen mask was 0x0800, the color intensity bit (controlling brightness of the foreground) would be preserved. Hence, the mouse cursor would become bright of it passed throught an area of bright text. Or, if the screen mask was 0xF000, and the background color of the cursor mask was 0 (ie in the form 0x0---), the current background color will always be maintained underneath the mouse cursor. The effects become very strange if you use nonzero values in the character portion of the screen mask (the last two digits), which causes the screen characte to affect the look of the mouse cursor. This is best left t private experimentation.

Return Value:

None.

See Also:

TC() macro Set_Mouse_Graphic_Cursor

Set_Mouse_Graphic_Cursor

Syntax:

```
     void Set_Mouse_Graphic_Cursor(mouse_t Hot_X, mouse_t Hot_Y,
                                   mouse_t Cursor[2][16]);
```

Description:

Set_Mouse_Graphic_Cursor changes how the mouse cursor will appear on the screen, while in graphic modes. The graphic cursor can be used in any of the graphic modes. The graphic cursor is defined by two 16 bit by 16 bit arrays, the screen mask, and the cursor mask. This defines a 16 by 16 pixel square in high-resolution or EGA mode, an 8 by 16 pixel block in medium resolution 4-color mode, and a 4 by 16 pixel block in low resolution 16 color mode.

When a graphic cursor is displayed on the screen, three operations take place. First, the screen image "under the cursor is saved, Then, the screen mask is logically ANDe with the screen image. Finally, the cursor mask is logically XORed with the result of the first operation.

The logically result of these operations is:

Screen mask	Cursor Mask	Result
0	0	0
0	1	1
1	0	Same as original bit
1	1	Inverse of original bit.

Remember, in CGA color modes, more than one bit is required for each pixel.

Return Value:

None

See Also:

Set_Mouse_Text_Cursor Peace[][] CrossHair[][] Lightening[

```
Example:
     #include <moustrap.h>
     #include <stdio.h>

     mouse_t Lightening[2][16] = {
          {
          0xFFF7,         /* 11111111 11110111b    */
          0xFFCF,         /* 11111111 11001111b    */
          0xFF9F,         /* 11111111 10011111b    */
          0xFF3F,         /* 11111111 00111111b    */
          0xFE7F,         /* 11111110 01111111b    */
          0xFCFF,         /* 11111100 11111111b    */
          0xF9FF,         /* 11111001 11111111b    */
          0xF803,         /* 11111000 00000111b    */
          0xFFE3,         /* 11111111 11100111b    */
          0xFFCF,         /* 11111111 11001111b    */
          0xFF9F,         /* 11111111 10011111b    */
          0xFF3F,         /* 11111111 00111111b    */
          0xFE7F,         /* 11111110 01111111b    */
          0xFCFF,         /* 11111100 11111111b    */
          0xF9FF,         /* 11111001 11111111b    */
          0xF7FF},        /* 11110111 11111111b    */

          {0x0008,         /* 00000000 00001000b    */
           0x0030,         /* 00000000 00110000b    */
           0x0060,         /* 00000000 01100000b    */
           0x00C0,         /* 00000000 11000000b    */
           0x0180,         /* 00000001 10000000b    */
           0x0300,         /* 00000011 00000000b    */
           0x0600,         /* 00000110 00000000b    */
           0x03F8,         /* 00000111 11111000b    */
           0x0018,         /* 00000000 00011000b    */
           0x0030,         /* 00000000 00110000b    */
           0x0060,         /* 00000000 01100000b    */
           0x00C0,         /* 00000000 11000000b    */
           0x0180,         /* 00000001 10000000b    */
           0x0300,         /* 00000011 00000000b    */
           0x0600,         /* 00000110 00000000b    */
           0x0800}         /* 00001000 00000000b    */
           };

     main()
     {
          mouse_t s,x,y;

          if (Check_Mouse()) {
               Set_Mouse_Graphic_Cursor(4,15,Lightening);
               Show_Mouse();
               }
     }
```

Chapter 3
Advanced Mouse Control Functions

These thirteen functions simplify the process of interpretin the users input using a mouse. They work on the assumption that most of the time a mouse is used by "clicking" a specific button at a specific place on the screen. They were written using Microsoft's C v5.1.

```
Activate_Mouse_Page()
Add_Mouse_Button()
Add_Mouse_Hot_Spot()
Add_Mouse_Page()
Clear_All_Mouse_Definition()
Clear_Mouse_Pages()
DeActivate_Mouse_Page()
Define_Mouse_System()
Delete_Mouse_Button()
Delete_Mouse_Hot_Spot()
Delete_Mouse_Page()
Get_Char_Mouse_Kbd()
Read_Mouse()
Read_Mouse_Kbd()
```

Activate_Mouse_Page

Syntax:

```
     #include <moustrap.h>
     mouse_t Activate_Mouse_Page(mouse_t Page_ID)
```

Description:

The Activate_Mouse_Page function sets active one of the previously defined mouse pages. In Single Page mode, the currently active page is cleared, and the mouse cursor is limited to the area of that page. In Overlaid mode, the curr pages remain active, and the mouse cursor area is widened, i necessary, to accommodate the new page.

Return Value:

MNOERROR if there was no problem, otherwise
MERROR with M_Error set to the specific error.

See Also:

M_Error, DeActivate_Mouse_Page, Add_Mouse_Page

Example:

See Chapter 4.

Add_Mouse_Page

Syntax:

```
     #include <moustrap.h>

     mouse_t Add_Mouse_Page(Page_Type, Top, Left, Bottom, Right);

     mouse_t         Page_Type;      /* M_Text_Coord   or  */
                                     /* M_Graphic_Coord    */
     mouse_t         Top;
     mouse_t         Left;           /* Coordinates of corners */
     mouse_t         Bottom;         /* of the page.           */
     mouse_t         Right;
```

Description:

Defines a new mouse page which is added to the system. Page_Type tells if the corner points are given usin text coordinates (80x25) or Graphic coordinates (640x200). Coordinates of Hot Spots for this page are also assumed to be given using that system.

Return Value:

Returns a Page ID number, which is to be used to reference this page in the future, or, MERROR if there was problem, with it's cause given in M_Error.

See Also:

Delete_Mouse_Page, Add_Mouse_Button, Add_Mouse_Hot_Spot
M_Error, Activate_Mouse_Page, DeActivate_Mouse_Page

Example:

See Chapter 4.

Add_Mouse_Button

Syntax:

```
     #include <moustrap.h>

     mouse_t Add_Mouse_Button(Page_ID, Button, Return_Value);
     mouse_t Page_ID;
     mouse_t Button;
     mouse_t Return_Value;
```

Description:

Add_Mouse_Button lets you tell the system how to react to a certain button being pressed. Page_ID is the page which this definition refers to, or if 0, the definition is valid in all pages. Button is either M_Left, M_Right, or M_Center. If the Return_Value is 0, it's assumed the Hot Spots are associated with this button in this page. Otherwis the Return_Value is any value the user wished to assign. It's return by Read_Mouse and Get_Char_Mouse_Kbd if that button is pressed while that page is active. In Overlaid mode, if more than one page, with conflicting definitions, are active the most recent Add_Mouse_Button has precedence. Any definition of a particular Page/Button combination replaces any previous definition of that combination.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also

M_Error, Add_Mouse_Page, Add_Mouse_Hot_Spot

Example:

See Chapter 4.

Add_Mouse_Hot_Spot

Syntax:

```
#include <moustrap.h>

mouse_t Add_Mouse_Hot_Spot(Page_ID, Button, Top, Left,
                       Bottom, Right, Return_Value);
mouse_t Page_ID;
mouse_t Button;
mouse_t Top;            /* corner of the area */
mouse_t Left;
mouse_t Bottom;
mouse_t Right;
mouse_t Return_Value;
```

Description:

Add_Mouse_Hot_Spot defines an area such that if the appropriate Button is pressed while the mouse cursor is within the area given while the page given by Page_ID is active, Read_Mouse will return Return_Value. A maximum of 65535 hot spots can be defined.

Return Value:

An ID number for this hot spot, if there was no problem; otherwise MERROR with the specific error given in M

See Also:

M_Error, Delete_Mouse_Hot_Spot

Example:

See Chapter 4.

Clear_All_Mouse_Definitions

Syntax:

```
     #include <moustrap.h>

     mouse_t Clear_All_Mouse_Definitions(void);
```

Description:

Erases everything. Removes all Page, Button, and Hot Spot definitions. Reset various internal variables. Must be done before switching between Single Page & Overlaid modes.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also:

M_Error, Define_Mouse_System

Example:

See Chapter 4.

Clear_Mouse_Pages

Syntax:

```
     #include <moustrap.h>

     mouse_t Clear_Mouse_Pages(void);
```

Description:

Deactivates all mouse pages. Hides cursors. Resets cursor limits.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also:

M_Error, DeActivate_Mouse_Page, Activate_Mouse_Cursor

Example:

See Chapter 4.

DeActivate_Mouse_Page

Syntax:

```
     #include <moustrap.h>

     mouse_t DeActivate_Mouse_Page(Page_ID);
     mouse_t Page_ID;
```

Description:

Deactivates the referenced mouse page. Button and Hot Spot definitions linked to that page will no longer function unti restarted with Activate_Mouse_Page. On Single page mode, this is done automatically when another page is activated.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also:

M_Error Activate_Mouse_Page, Clear_Mouse_Pages

Example:

See Chapter 4.

Define_Mouse_System

Syntax:

```
#include <moustrap.h>

mouse_t Define_Mouse_System(Page_Type);
mouse_t Page_type;
```

Description:

Define_Mouse_System declares how mouse pages are to be used through the program. Page_Type must be either M_Overlaid_Pa or M_Single_Pages. Automatically initializes mouse by executing Check_Mouse. Can only be done once in a program unless reset with Clear_All_Mouse_Definitions.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also:

M_Error, Clear_All_Mouse_Definitions, Check_Mouse

Example:

See Chapter 4.

Delete_Mouse_Button
Delete_Mouse_Hot_Spot
Delete_Mouse_Page

Syntax:

```
     #include <moustrap.h>

     mouse_t Delete_Mouse_Button(mouse_t Page_ID, mouse_t Button)
     mouse_t Delete_Mouse_Hot_Spot(mouse_t HS_ID);
     mouse_t Delete_Mouse_Page(mouse_t Page_ID);
```

Description:

Removes the indicated item from the system.

Return Value:

MNOERROR if there was no problem; otherwise
MERROR with the specific error given in M_Error

See Also:

M_Error

Example:

See Chapter 4.

Get_Char_Mouse_Kbd

Syntax:

```
#include <moustrap.h>

mouse_t Get_Char_Mouse_Kbd(void);
```

Description

Get_Char_Mouse_Kbd acts much like the standard library function GETCH, but will accept input from either the keyboard or the mouse. Will return only when some input is received from the keyboard or mouse.

Return Value:

The value inputted if there was no problem; otherwise MERROR with the specific error given in M_Error

See Also:

M_Error, Read_Mouse

Example:

See Chapter 4.

Read_Mouse

Syntax:

```
#include <moustrap.h>

mouse_t Read_Mouse(void)
```

Description:

Checks mouse for input.

Return Value:

The Return_value assigned to a Button or Hot Spot, if that item was "clicked" on, or
MERROR if an error occured, or
0 if no button was pressed.

See Also:

M_Error, Get_Char_Mouse_Kbd

Example:

See Chapter 4.

NOTE:

Remember, a non-zero return value does not necessarily mean a button was pressed; it could also indicate an error condition. (MNOINIT or MNOACTIVE).

Read_Mouse_Kbd

Syntax:

```
#include <moustrap.h>

mouse_t Read_Mouse_Kbd(void)
```

Description:

Something between Read_Mouse and Get_Mouse_Kbd_Char. Checks mouse and the keyboard for input.

Return Value:

The Return_value assigned to a Button or Hot Spot, if that item was "clicked" on, or
a key pressed on the keyboard.
MERROR if an error occured, or
0 if no button or key was pressed.

See Also:

M_Error, Get_Char_Mouse_Kbd

Example:

See Chapter 4.

NOTE:

Remember, a non-zero return value does not necessarily mean a button was pressed; it could also indicate an error condition. (MNOINIT or MNOACTIVE).

Chapter 4

Using the MouseTrap Library

The basic concept of the MouseTrap is the "Mouse Page". A Mouse Page is one set of button and "Hot spot" definitions. Any character can be assigned to a button or hot spot. Pages can be used in either of two ways: You can have up ot 65,000 single pages, which can only be used one at a time, or up to 16 page "overlaid" pages, any combination of which can be active at once. You choose this by using the Define_Mouse_System function, with either M_Single_Pages or M_Overlaid_Pages.

The next step is to define a page, by using the Add_Mouse_Page function, passing to it the "type" of page it is, either M_Graphic_Coo or M_Text_Coord; and the 4 corner points for that page using the appropriate set of coordinates (either 80x25 or 640x200). Using "0" for each corner will have it using the entire screen. Add_Mouse_Page will return an ID number for the page, which you will be using in all references to this page.

Next, you must define the buttons you will be using. This is don with Add_Mouse_Button. You tell it which for which page and button this definition is to apply, and the value to return if that button wa clicked while that page was active. If you use "0" for the Page ID, this definition will apply to all pages. If you use "0" for the return value, you can have that button return different values for being clicked at different "hot spots" within the page.

If you are using hot spots in a page, you must next call Add_Mous _Hot_Spot, passing to it the page ID and button code, the corner points, and the return value for the spot.

Now, we get to the fun part. Choose a page using the Activate- _Mouse_Page function. Using overlaid page, you can have several pages active at once; remove them with the DeActivate_Mouse_Page or Clear_Mouse_Pages function. In single page mode, activating a new pag automatically deactivates the last one.

Finally, simply call Read_Mouse(). It will return either the value for the button or Hot spot clicked or 0 if no button was clicked. Or simpler still, use Get_Char_Mouse_Kbd(), which waits until some input is entered by either keyboard or mouse.

To further exemplify the process let's examine the sample program MICETEST.C:

```
#include <stdio.h>
#include "moustrap.h"

#include <graph.h>

main ()
{
        mouse_t y,z,c;
```

The data type "mouse_t" is defined in MOUSTRAP.H. All variables used with the MouseTrap library should be define as this type.

The first group of lines setup the screen so it's easier to understand what's happening with the mouse. But by themselves they do nothing of interest to this discussion. Ignore them and skip down bit.

```
        Define_Mouse_System(M_Single_Pages);
```

For the first step, we going to be using single pages; only one of the pages we're about the define can be active at only given time.

```
        y=Add_Mouse_Page(M_Text_Coord,15,20,24,40);

        z=Add_Mouse_Page(M_Text_Coord,5,10,15,20);
```

Next, we define two mouse pages, Y & Z. Y is limited to the rectangle from row 15, column 20 to row 24, column 40. Similarly, Z is the area from (5,10) to (15,20).

```
        Set_Mouse_Text_Cursor(0,0,TC(' ',4,4));
```

Now, we describe how the mouse cursor will look. We start with something simple. We'll use a software cursor, with no screen mask. We use the macro TC(), defined in MOUSTRAP.H, to build a cursor which is just a space, with a red foreground (color 4) on a red background.

```
        Add_Mouse_Button(0,M_Middle,'2');
```

For our first button definition, we'll say that anytime the Middle button is pressed, Read_Mouse will return an ASCII character '2', regardless of what page is active (provided at least one page IS active).

```
        Add_Mouse_Button(z,M_Left,'1');
```

Next we'll have pressing the Left button return an ASCII '1' whenever page Z is active.

```
        Add_Mouse_Button(z,M_Right,0);
        Add_Mouse_Hot_Spot(z,M_Right,7,13,13,18,'C');
```

Now, we add our first Hot Spot. Here, we must first declare the Right button in Page Z, then we declare the area in the rectangle (7,13) - (13,18) as a hot spot returning the character 'C' when that button is pressed while page Z is active. Since no other hot spot is defined, clicking the right button outside that area will return 0, just as if no click had occurred.

```
        Add_Mouse_Button(y,M_Left,0);

        Add_Mouse_Hot_Spot(y,M_Left,15,20,24,30,'L');
        Add_Mouse_Hot_Spot(y,M_Left,15,30,24,40,'R');
```

Continuing in a similar vein, we define two hot spots in page Y. When the Left button is pressed, if the cursor is in the left side we'll get the character 'L', while the right side return the character 'R'.

```
do {

        Activate_Mouse_Page(z);
```

As we enter the loop, we activate page Z. The mouse cursor is "turned on" with it's movement limited to the edges of the page.

```
        c=Get_Char_Mouse_Kbd();
        printf("Page Z: Character \"%c\"",c);
```

We stop, and get a character from either the keyboard (which is not much fun), or via the mouse; and print it.

```
        Activate_Mouse_Page(y);
        c=Get_Char_Mouse_Kbd();
        printf("Page Y: Character \"%c\"",c);
```

Now, we activate page Y (which automatically deactivates page Z). The mouse cursor moves into the new area, and it's motion is limited to that range. We get another character and print it.

```
        } while (c!='Q');

        Clear_All_Mouse_Definitions();

        Define_Mouse_System(M_Overlaid_Pages);
```

Now, we want to start over, so we clear all the old definitions, and restart, but this time using overlaid pages.

```
        y=Add_Mouse_Page(M_Text_Coord,15,20,24,40);
        z=Add_Mouse_Page(M_Text_Coord,5,10,15,20);
```

```
        Add_Mouse_Button(0,M_Middle,'2');
        Add_Mouse_Button(z,M_Left,'1');
        Add_Mouse_Button(z,M_Right,0);
        Add_Mouse_Hot_Spot(z,M_Right,7,13,13,18,'C');
        Add_Mouse_Button(y,M_Left,0);
        Add_Mouse_Hot_Spot(y,M_Left,15,20,24,30,'L');
        Add_Mouse_Hot_Spot(y,M_Left,15,30,24,40,'R');

        Set_Mouse_Text_Cursor(0,0,TC('+',4,2));
```

We'll redefine all of our pages, buttons, and hot spots, exactly as we did the first time. We'll also change the mouse cursor, this time to something a bit more exciting than before, a red plus sign on a green background (color 2).

```
do {
        Activate_Mouse_Page(z);
        c=Get_Char_Mouse_Kbd();
        printf("Page Z: Character \"%c\"",c);
```

Again, we activate page Z, and get a character from it. This works exactly as it did in the first loop using single page mode.

```
        DeActivate_Mouse_Page(z);
        Activate_Mouse_Page(y);
        c=Get_Char_Mouse_Kbd();
        printf("Page Y: Character \"%c\"",c);
```

And again, we activate page Y, and get a character from it. The only difference is that we had to first deactivate page Z.

```
        Activate_Mouse_Page(z);
        c=Get_Char_Mouse_Kbd();
        printf("Page Y & Z: Character \"%c\"",c);
```

Now we get flashy. Without deactivate page Y, we'll reactivate page Z. You will notice that the mouse can now move within a much larger area, specifically the rectangle which circumscribes both of the smaller rectangle. Notice that if you click the left button in the area that is not within the boundaries of either page, "1" will be return. This is because the button definition says to return "1" whenever page Z is active, regardless of where the cursor is, even if it is outside the stated area of Page Z. However, notice that the Hot Spots of the left button in page Y, take precedence over this. This is because the BUTTON definition on page Y was give after the button definition off page Z, and therefore overrules it.

```
        DeActivate_Mouse_Page(y);
```

Now, we deactivate page Y, leaving only page Z. Notice that the mouse's movements are once again restricted to the area of page Z. (See..I told you that I would fix this problem with release 1.01)

```
          }
}
```

You should also remember that, although we always used ASCII characters for return values in the example, ANY character or integer value, in the range of 1 to 65534, can be used.

Chapter 5

Technical Specification

This chapter is provided only to those who have paid to become registered uses. This was done because I assumed that it would be of little use to anyone who didn't have the source code (which also comes with registration). This method also gives me a few extra weeks to write it.

Chapter 6

Appendix

A.   MOUSTRAP.H
B.   Global Variables
C.   Error Codes
D.   Reference
E.   Support
F.   Release History

Appendix A : MOUSTRAP.H

The header file, MOUSTRAP.H should be included in every C program which uses the MouseTrap Library functions. It includes complete function prototypes for each of the MouseTrap Library functio In addition, it defines a number of constants which are to be used with the functions. These include:

M_Overlaid_Pages -and- M_Single_Pages, which are used with the function Define_Mouse_System.

M_Text_Coord -and- M_Graphic_Coord, which are used with the Add_Mouse_Page function, to tell which system screen coordinates are being given in.

M_Left, M_Right, M_Center, -and- M_Middle, which are used to refer to the mouse buttons whenever necessary. Note that M_Center and M_Middle are equivalent, and you may use whichever holds your fancy.

M_HORIZ -and- M_VERT, which are used with Set_Mouse_Limits.

MERROR -and- MNOERROR, (-1 and 0, respectively), which are return by various functions to indicate whether an error occurred. Also defined are a large number of error codes, which are discussed further in appendix C.

The macro TC() is used to create an integer value in the for BFCC, where B is the background color, F, the foreground color, and CC is a character. This value is used by Set_Mouse_Text_Curs (and by other routines outside of the MouseTrap Library which perform direct screen writes). The macro require that yo give it the character, foreground and background color code. Th statement TC('A',14,5) would produce the code 5E41h, which mean the letter 'A' in bright Yellow on a Magenta background.

It also declared three global variables, which are described in Appendix B.

Also defined is the data type "mouse_t" which is used to define vitually every variable used in the MouseTrap functions. Also included are the structure definitions for Pages, buttons, and hot spots. I'll not should what you can use them for, but I thought you might be interested.

The last group of lines will"force" LINK to include the proper version of the MouseTrap library, without being explicitly told to. This will only work with Microsoft C 5.1. Other compliers will probable generate a warning for these lines.

Appendix B : Global Variables

There are three global variables used with the MouseTrap Function. They are:

_mouse_there: This is initialized to 0, meaning no mouse available, and is set by calling either Check_Mouse or Define_Mouse_System. After a call to either of those functions it is set to either 0, meaning that there is STILL no mouse on this system, or, 2 or 3, giving the number of buttons on the mouse. Since "no mouse" is zero, and "mouse present" is nonzero, this variable can also be used as a TRUE/FALSE value.

M_Paging_Method: This simply holding the value you used with Define_Mouse_System, and is either M_Overlaid_Pages or M_Single_Pages.

M_Error: This holds the error code of the last error that occurre Full description of the error codes is given in Appendix C.

And, there are three predefined graphic cursors:

Lightening	:	A Lightening Bolt, use hot spot
PeaceSign	:	A peace sign. use hot spot.
CrossHair	:	A cross hair, use hot spot

Appendix C : Error Codes.

Error conditions are indicated by a function returning the value MERROR (-1) with the error code given in the global variable M_Error. The error code remains in M_Error until either cleared manually by the user or altered by another error.

The error code are:

MNOINIT An attempt was made to use one of the advanced fu without first calling Define_Mouse_System

NOMOUSE An attempt was made to use a function while no mouse was attached to the system.

NOSPACE An attempt to add a new Page, Button, or hot spot failed because there was not enough availabl RAM. Since these definitions use so little memory, this error should rarely occur.

MTOOMANY An attempt was made to define more than 16 pages in Overlaid mode, or more than 65536 pages in single page mode, or more than 65536 hot spots (in either mode). Remember, that all hot spots defined, even those later deleted, count towards this limit. (Deleted pages, however, do not.)

MNOREINIT An attempt was made to call Define_Mouse_System, after pages were added. All pages (and buttons, and hot spots) must be removed before a second call to Define_Mouse_System may be made. Use Clear_All_Mouse_Definitions().

MNOTPAGE An attempt was made to reference a page which had not yet been defined.

MNOTBUTTON An attempt was made to reference a button which ha not yet been defined.

MNOTHOTSPOT An attempt was made to reference a hot spot which had not yet been defined.

MBUTTONRET An attempt was made to tied a hot spot to button which already has it's own return value.

MNOACTIVE An attempt was made to call Read_Mouse with no pag active.

NOTE: The MouseTrap library does not display any form of error message. The only way it alerts you that it feels somethin has gone awry is throught these flags.

Appendix D : Reference

Much of the information used to create this manual was taken from

Mouse System Corporation, Optimouse Reference Manual, version 4.0, Copyright 1984, 1985.

Cort, Nigel. "How to Handle a Mouse, part 1." The C Gazette. 2:4, March 1988.

Cort, Nigel. "How to Handle a Mouse, part 2." The C Gazette. 3:1, Summer 1988.

Appendix E : Support

James M. Curran is the author of the MouseTrap Functions and is solely responsible for it's content. Any comments, problem, suggestions, marriage proposals, or death threats stemming from this library should be directed to him at:

James M. Curran
24 Greendale Road
Cedar Grove, NJ 07009-1313

Don't forget the "M." since the family is just swarming with "James Curran"'s

He can also be reached via Compuserve at [72261,655].

And for the more adventurous, he's also a regular on several northern New Jersey BBS's under the handle "The Perfect Stranger"

Special Note for version 1.0: Because of the painfully short time between the original inspiration for this, and my leaving on a Europea trip (which, I assume will do much to help me forget said pain), this entire project was written, debugged, and documented in 10 days (while still working at the day job). Obviously, there HAS to be some bugs lurking out there somewhere, if not in the functions themselves, in this documentation (there are, by the way, three functions in the library that are not described here). Any feedback from users will be of great assistance in putting out version 1.1 when I get back.

Appendix F : Revision History

9/21/88: Original Release!

10/22/88: Revision 1.01: Corrected typo's and clarified areas of the documentation. Rewrote the Set_Mouse_Limits function Finally documented Get_Mickeys, and Set_Mouse_Graphic_Curs and added the sample cursors (Peace, Lightening, CrossHair). Eventually wrote Chapter 5.

2/89 : Converted documentation to WordPerfect. (May actually be updated in a timely fashion now) Corrected Table of Contents and added Index, and did all kinds of things one can do when I finally gets a REAL word processor. (Note how I carefull avoid slandering my last word processor).

Appendix I : Index

Activate_Mouse_Page() 1

Add_Mouse_Button() 1

Add_Mouse_Hot_Spot() 1

Add_Mouse_Page() 1

Check_Mouse() 4-7, 9-11, 1

Clear_All_Mouse_Definition() 1

Clear_Mouse_Pages() 1

DeActivate_Mouse_Page() 1

Define_Mouse_System() 5, 1

Delete_Mouse_Button() 1

Delete_Mouse_Hot_Spot() 1

Delete_Mouse_Page() 1

Functions 4

Get_Char_Mouse_Kbd() 16, 29, 31, 3

Get_Mouse_Position()

Get_Mouse_Press()

Get_Mouse_Release()

Hide_Mouse() 4,

Read_Mouse() 1

Set_Mouse_Limits()

Set_Mouse_Position()

Set_Mouse_Text_Cursor()

Show_Mouse() 4, 6,