Chapter 4

Mouse Interface Routines

CMX11 includes several mouse interface routines that are callable from both C* and CM Fortran. From C*, most of them are passed a pointer type called **CMXMousePoint_t**. In CM Fortran, this looks like an integer. **CMXMouse-Point_t** is defined as:

```
typedef struct
{
    int x, y, button;
}
_CMXMousePoint, *CMXMousePoint_t;
```

4.1 Mouse Interaction Routines

Four CMX11 routines supply the basic functionality of the mouse interface at a high level:

- **CMXCurrentMousePoint**
- **CMXGetMousePoint**
- **CMXGetMouseLine**
- **CMXGetMouseRectangle**

CMXCurrentMousePoint queries the state of the physical mouse in the X server and fills in the mouse point structure with the current *x* and *y* coordinates of the mouse, along with its buttons.

Copyright © 1993 Thinking Machines Corporation

From CM Fortran:

```
SUBROUTINE CMXCurrentMousePoint(widget, point)

INTEGER widget
INTEGER point
```

From C*:

```
void CMXCurrentMousePoint(Widget widget,
                          CMXMousePoint_t point);
```

In *point*, *x* and *y* are the current coordinates, relative to the window, *not* to the screen as a whole. They might be negative or larger than the window size if the pointer is not within the window.

The button member of the mouse point is filled in with the current state of the mouse, which is the OR of the following constants: **ShiftMask**, **LockMask**, **ControlMask**, **Mod1Mask**, **Mod2Mask**, **Mod3Mask**, **Mod4Mask**, **Mod5Mask**, **Button1Mask**, **Button2Mask**, **Button3Mask**, **Button4Mask**, and **Button5Mask**, depending on whether the appropriate button or key is pressed. These constants are defined by including **cmx-cmf.h** for CM Fortran and **Xlib.h** for C*.

The other routines, **CMXGetMousePoint**, **CMXGetMouseLine**, and **CMXGetMouseRectangle**, wait for a button event to be triggered on the mouse. They then return an **INTEGER** identifying the point data object containing information on the mouse point's location at the time of the event and the button that caused it.

CMXGetMousePoint returns the location of the mouse point when a button is pressed:

From CM Fortran:

```
SUBROUTINE CMXGetMousePoint(widget, point)

INTEGER widget
INTEGER point
```

From C*:

```
void CMXGetMousePoint(Widget widget,
                      CMXMousePoint_t point);
```


Copyright © 1993 Thinking Machines Corporation

CMXGetMousePoint waits for a button to be pressed and released, and then returns the coordinates of the release in *point.* The button member of the point is filled in with one of the coordinates **Button1**, **Button2**, or **Button3**, depending on which button is pressed. These constants are defined by including **cmx-cmf.h** for CM Fortran and **Xlib.h** or **cm/cmxdisplay.h** for C*.

CMXGetMouseLine allows the user to define two points, the start and end points of a line, and returns the button information.

From CM Fortran:

```
SUBROUTINE CMXGetMouseLine(widget, p1, p2, anchorp)

INTEGER widget
INTEGER p1, p2
INTEGER anchorp
```

From C*:

```
void CMXGetMouseLine(Widget widget, CMXMousePoint_t p1,
                     CMXMousePoint_t p2, int anchorp);
```

CMXGetMouseLine draws a "rubber band" line. First it waits for a mouse button to be pressed; then it draws a moving line from that point to the current mouse point as the mouse is dragged, until the mouse button is released. **CMXGetMouseLine** returns the start coordinate in *p1* and the end coordinate in *p2*.

If *anchorp* is true (nonzero), the first point is "anchored" to the contents of *p1*. In other words, *p1* is used as an input *and* output parameter, and the point where the button is pressed is insignificant; only the release point is used. You can use **CMXSetMousePointLocation** to set *p1* to an initial value.

The button members of *p1* and *p2* are filled in with **Button1**, **Button2**, or **Button3**, depending on which button is pressed and released.

Similarly, **CMXGetMouseRectangle** allows you to define a rectangle, and returns the button information.

Copyright © 1993 Thinking Machines Corporation

From CM Fortran:

```
SUBROUTINE CMXGetMouseRectangle(widget, p1, p2, anchorp)

INTEGER widget
INTEGER p1, p2
INTEGER anchorp
```

From C*:

```
void CMXGetMouseRectangle(Widget widget,
         CMXMousePoint_t p1, CMXMousePoint_t p2,
         int anchorp);
```

CMXGetMouseRectangle draws a "rubber band" rectangle. One corner is defined by the coordinates of the mouse when a button is pressed (or, if *anchorp* is true, *p1*); the opposite corner is defined by the position of the mouse, until the button is released. **CMXGetMouseRectangle** returns the start coordinate in *p1* and the end coordinate in *p2*.

The button members of *p1* and *p2* are filled in with **Button1**, **Button2**, or **Button3**, depending on which button is pressed and released.

4.2 Allocating the Mouse Point Object

The mouse point object is passed to or returned from the following CMX11 functions:

- **CMXGetMousePoint**
- **CMXGetMouseLine**
- **CMXGetMouseRectangle**
- **CMXMousePointX**
- **CMXMousePointY**
- **CMXMousePointButton**
- **CMXGetMousePoint**
- **CMXCurrentMousePoint**

Copyright © 1993 Thinking Machines Corporation

The library provides routines, **CMXAllocateMousePoint** and **CMXDeallocateMousePoint**, that allow you to allocate and deallocate the mouse point object.

From CM Fortran:

```
INTEGER FUNCTION CMXAllocateMousePoint( )

SUBROUTINE CMXDeallocateMousePoint(point)
INTEGER point
```

From C*:

```
CMXMousePoint_t CMXAllocateMousePoint(void);

void CMXDeallocateMousePoint(CMXMousePoint_t point);
```

You can set the initial mouse point position in a mouse point object by calling **CMXSetMousePointLocation**.

From CM Fortran:

```
SUBROUTINE CMXSetMousePointLocation(point, x, y)

INTEGER point
INTEGER x, y
```

From C*:

```
void CMXSetMousePointLocation(CMXMousePoint_t point,
                              int x, int y);
```

Notice that **CMXSetMousePointLocation** does not change the location of the mouse directly, but only sets the coordinates in the mouse point object. These coordinates take effect only when another routine references the mouse point object.

4.3 Accessing Mouse Point Information

Your application can read the coordinate and button information in a mouse point data object by using the point accessor routines.

Copyright © 1993 Thinking Machines Corporation

CMXMousePointX and **CMXMousePointY** return the current mouse point *x* and *y* coordinates from the mouse point object.

From CM Fortran:

```
INTEGER FUNCTION CMXMousePointX(point)
INTEGER point

INTEGER FUNCTION CMXMousePointY(point)
INTEGER point
```

From C*:

```
int CMXMousePointX(CMXMousePoint_t point);

int CMXMousePointY(CMXMousePoint_t point);
```

CMXMousePointButton returns the button constant for the last button pressed or released.

From CM Fortran:

```
INTEGER FUNCTION CMXMousePointButton(point)
INTEGER point
```

From C*:

```
int CMXMousePointButton(CMXMousePoint_t point);
```

After a call to **CMXGetMousePoint**, **CMXGetMouseLine**, or **CMXGetMouseRectangle**, the button member contains one of **Button1**, **Button2**, or **Button3**, depending on the button that was last pressed or released.

To check which button was last pressed or released, compare the return value from these routines with the constant. For example, to test whether the reason for the event was the first button on the mouse being pressed, use the following conditional:

In CM Fortran:

```
IF (CMXMousePointButton(p) .EQ. BUTTON1)
```

In C*:

```
if (CMXMousePointButton(p) == Button1)
```


Version 1.6, October 1993
Copyright © 1993 Thinking Machines Corporation

After a call to **CMXCurrentMousePoint**, the button member is the logical OR of any or all of the following constants: **ShiftMask**, **LockMask**, **ControlMask**, **Mod1Mask**, **Mod2Mask**, **Mod3Mask**, **Mod4Mask**, **Mod5Mask**, **Button1Mask**, **Button2Mask**, **Button3Mask**, **Button4Mask**, or **Button5Mask**, depending on which buttons or keys were pressed.

As with the button constants, you can determine the state of the buttons by comparing this return value with the constants, as in the following code examples.

In CM Fortran:

```
C    IS ANY BUTTON PRESSED?
      IF (CMXMousePointButton(p) .NE. 0)

C    ARE ALL BUTTONS RELEASED (UP)?
      IF (CMXMousePointButton(p) .EQ. 0)

C    IS BUTTON 1 PRESSED? (DON'T CARE ABOUT OTHER BUTTONS)
      IF ((IAND(CMXMousePointButton(p), BUTTON1MASK) .NE. 0)

C    IS ONLY BUTTON 1 PRESSED? (OTHER BUTTONS MUST BE UP)
      IF (CMXMousePointButton(p) .EQ. BUTTON1MASK)

C    ARE BOTH BUTTON 1 AND BUTTON 2 PRESSED?
      IF ((IAND(CMXMousePointButton(p), BUTTON1MASK) .NE. 0)
     & .AND.
     & (IAND(CMXMousePointButton(p), BUTTON2MASK) .NE. 0))

C    IS EITHER BUTTON 1 OR BUTTON 2 PRESSED?
      IF ((IAND(CMXMousePointButton(p),
     & IOR(BUTTON1MASK, BUTTON2MASK))) .NE. 0)

C    IS BUTTON 2 NO LONGER PRESSED (I.E., IT'S UP)?
      IF (IAND(CMXMousePointButton(p), BUTTON2MASK) .EQ. 0)
```

In C*:

```
/* Is any button pressed? */
if (CMXMousePointButton(p) != 0)

/* Are all buttons released (up)? */
if (CMXMousePointButton(p) == 0)

/* Is button 1 pressed? (don't care about other buttons) */
if ((CMXMousePointButton(p) & Button1Mask) != 0)
```


Copyright © 1993 Thinking Machines Corporation

```
/* Is ONLY button 1 pressed? (other buttons must be up) */
if (CMXMousePointButton(p) == Button1Mask)

/* Are both button 1 and button 2 pressed? */
if (((CMXMousePointButton(p) & Button1Mask) != 0)
&& ((CMXMousePointButton(p) & Button2Mask) != 0))

/* Is either button 1 or button 2 pressed? */
if ((CMXMousePointButton(p) & (Button1Mask | Button2Mask))
== (Button1Mask | Button2Mask))

/* Is button 2 no longer pressed (i.e., it's up)? */
if (CMXMousePointButton(p) & Button1Mask == 0)
```


Copyright © 1993 Thinking Machines Corporation