EVENT QUEUE LIBRARY

for Turbo C and DJGPP

User's Manual

Written by:

Csaba Biegl
May, 1992

Abstract

This document describes an event queue library which can be used in Turbo C and DJGPP programs. The event queue mechanism is interrupt driven and the queue stores keyboard and mouse events in the order of their arrival. In order for the mouse services of this library to work an MS compatible mouse driver has to be installed on the PC.

Introduction -- why event queues?

The "standard" MS-DOS programmer's interface to the keyboard (via DOS or BIOS calls) and the mouse (via the mouse driver) has a couple of shortcomings -- especially for graphical user interfaces. These are:

- No mouse event history: the mouse driver provides some services for the application program to poll the mouse position, button states, etc.. These services can be adequate if the application can afford to spend most of its time in a tight loop waiting for these events. (For example a graphical editor) However, in applications which do lengthy computations without user interaction this method can easily lead to the loss of user interaction events.

- Limited mouse cursor drawing capabilities: the mouse driver does not support cursor drawing in extended, higher-resolution graphics modes. The solution is to have the application to draw the cursor after polling the mouse driver for the current cursor position. However, in programs which spend a lot of time not waiting for user input it will lead to long periods when the cursor is frozen.

- Limited keyboard buffer size. Unless reprogrammed by the application the standard BIOS keyboard buffer size is only 16 bytes, which may be insufficient for longer type-ahead sequences.

- Missing keyboard modifier keys in the BIOS buffer: the standard BIOS keyboard services cannot determine the status of keyboard modifier keys for previously entered keys. (It is possible to obtain these values at the moment the keystroke is received.)

The interrupt driven event queue library described in this manual attempts to solve the above problems. It stores all user interaction events (key presses and mouse button changes) in a FIFO queue together with the status of the keyboard modifier keys and a time stamp of the event. The size of this queue is programmable by the application, thus the queue can be large enough to avoid event loss.

NOTE: although this event queue library is used in the LIBGRX graphics library, it is in no way attached to LIBGRX. The event queue mechanism can be used in any Turbo C or DJGPP program.

Data structures

The application program can obtain the user input events in an 'EventRecord' structure which is shown below (all of the following declarations and macros can be found in the include file "eventque.h"):

```
typedef struct {
    unsigned char   evt_type;            /* event type: 0: keyboard, 1: mouse */
    unsigned char   evt_kbstat;          /* keyboard status (ALT, SHIFT, etc..) */
    unsigned char   evt_mask;            /* mouse event mask */
    unsigned char   evt_button;          /* button status */
    unsigned short  evt_xpos;            /* X coord (or keycode if keybd event) */
    unsigned short  evt_ypos;            /* Y coord */
    unsigned long   evt_time;            /* time stamp of event */
#define evt_keycode evt_xpos             /* reuse this  slot for keybd events !! */
} EventRecord;
```

Various macros have been declared to help in decoding the values in the slots of the event record structure:

```
/*
 * event types
 */
#define EVENT_KEYBD      0
#define EVENT_MOUSE      1

/*
 * MOUSE event flag bits (values in the 'evt_mask' slot)
 * (also defined in "mousex.h" of the graphics library)
 */
#define M_MOTION         0x001
#define M_LEFT_DOWN      0x002
#define M_LEFT_UP        0x004
#define M_RIGHT_DOWN     0x008
#define M_RIGHT_UP       0x010
#define M_MIDDLE_DOWN    0x020
#define M_MIDDLE_UP      0x040
#define M_BUTTON_DOWN    (M_LEFT_DOWN | M_MIDDLE_DOWN | M_RIGHT_DOWN)
#define M_BUTTON_UP      (M_LEFT_UP   | M_MIDDLE_UP   | M_RIGHT_UP)
#define M_BUTTON_CHANGE  (M_BUTTON_UP | M_BUTTON_DOWN )

/*
 * MOUSE button status bits (in the 'evt_button' slot)
 */
#define M_LEFT           1
#define M_RIGHT          2
#define M_MIDDLE         4

/*
 * KEYBOARD status word bits ('evt_kbstat' slot)
 * (also defined in "mousex.h" of the graphics library)
 */
#define KB_RIGHTSHIFT    0x01            /* right shift key depressed */
#define KB_LEFTSHIFT     0x02            /* left shift key depressed */
#define KB_CTRL          0x04            /* CTRL depressed */
#define KB_ALT           0x08            /* ALT depressed */
#define KB_SCROLLOCK     0x10            /* SCROLL LOCK active */
#define KB_NUMLOCK       0x20            /* NUM LOCK active */
#define KB_CAPSLOCK      0x40            /* CAPS LOCK active */
#define KB_INSERT        0x80            /* INSERT state active */

#define KB_SHIFT         (KB_LEFTSHIFT | KB_RIGHTSHIFT)
```

The event queue is just a circular buffer of 'EventRecord' structures with a header structure as follows:

```
typedef struct {
    unsigned short  evq_maxsize;    /* max size of event queue */
    unsigned short  evq_cursize;    /* number of events in the queue */
    unsigned short  evq_rdptr;      /* next event to read */
    unsigned short  evq_wrptr;      /* next event to be written */
    short    evq_xpos;              /* current X coordinate of mouse */
    short    evq_ypos;              /* current Y coordinate of mouse */
    short    evq_xmin;              /* minimal mouse X coordinate */
    short    evq_ymin;              /* minimal mouse Y coordinate */
    short    evq_xmax;              /* maximal mouse X coordinate */
    short    evq_ymax;              /* maximal mouse Y coordinate */
    short    evq_xspeed;            /* horizontal speed (mickey/coord) */
    short    evq_yspeed;            /* vertical speed (mickey/coord) */
    unsigned short  evq_thresh;     /* fast movement threshold */
    unsigned short  evq_accel;      /* multiplier for fast move */
    unsigned char   evq_drawmouse;  /* interrupt handler has to draw mouse */
    unsigned char   evq_moved;      /* set if mouse moved */
    unsigned char   evq_delchar;    /* character removed from BIOS buffer */
    unsigned char   evq_enable;     /* event generation control flag */
    EventRecord     evq_events[1];  /* event buffer space */
} EventQueue;
```

The event queue circler buffer is allocated by a library function (see next section). Its size is determined at the time this init function is called. After initialization the application program receives a pointer to the queue header structure. NOTE: int the case of 32 bit DJGPP programs the event queue is allocated in the dos extender area, and a pointer into the low memory area is returned to the protected mode program.

The first four slots of the event queue header ('evq_maxsize' through 'evq_wrptr') **should never be changed by the application program**. These are for queue management and are used by the library internally. The remaining structure slots of the header can be used to control the mouse and the generation of events. The 'evq_xpos' and 'evq_ypos' slots contain the current mouse position. The 'evq_xmin' through 'evq_ymax' slots specify the mouse coordinate limits. The event queue library uses only the mouse mickey counters to calculate the cursor position. The reason for this is that some mouse drivers perform the rounding of mouse cursor coordinates to the nearest multiple of eight if they find themselves in a text or unrecognized graphics mode. The 'evq_xspeed' and 'evq_yspeed' slots control the speed of the mouse. The mouse mickey count changes are divided by these values before calculating the cursor position. The 'evq_thresh' and 'evq_accel' slots can be used to control the ballistic effect: if the cursor changes by more than the threshold the change is multiplied with the acceleration. NOTE: most mouse drivers already provide some amount of ballistic effect. The 'evq_drawmouse' flag controls whether the interrupt-driven mouse event handler updates the mouse cursor or not. The 'evq_moved' slot is set by the library whenever the mouse cursor position changes. An application might clear this slot and then monitor its value to detect a mouse movement.

The 'evq_delchar' slot affects the behavior of the keyboard event generation. If this slot is set then the key code is removed from the BIOS buffer, otherwise it is left there (even if an event entry is generated in the queue). The 'evq_enable' slot can be used to individually enable or disable the generation of mouse and keyboard events. The desired values for this slot can be obtained using the EVENT_ENABLE macro:

```
/*
 * set this bit in 'evq_enable' to enable the generation of the corresponding event
 * use EVENT_KEYBD or EVENT_MOUSE as argument
 */
#define EVENT_ENABLE(type)        (1 << (type))
```

Library functions

The library contains only the following three functions:

```
EventQueue *EventQueueInit(int qsize,int ms_stksize,void (*msdraw)(void));
void        EventQueueDeInit(void);
int         EventQueueNextEvent(EventQueue *q,EventRecord *e);
```

The 'EventQueueInit' function initializes the queue. The 'qsize' argument specifies the size of the circular buffer.

The 'msdraw' argument is the function which is used to update the mouse cursor position whenever it changes. The 'ms_stksize' argument specifies the size of the stack necessary for the cursor drawing function. If the cursor update function pointer is NULL the library will not attempt to update the cursor position. Otherwise the cursor update function should obtain the current mouse coordinates from the queue header 'evq_xpos' and 'evq_ypos'. If the 'evq_drawmouse' slot is cleared in the queue header then the mouse cursor drawing function will not be called. As mouse interrupts can occur at any time, the displaying and erasing of the mouse cursor should be done with some caution. For display: first the cursor should be drawn and 'evq_drawmouse' should be set only after this. For erase: first 'evq_drawmouse' should be cleared and the cursor erased next. The above sequence assures that no spurious mouse cursor drawing will happen. The 'evq_drawmouse' slot is cleared after the init function returns.

The 'EventQueueDeInit' function should be called before the termination of the program. **NOTE: it is very important to call this function before program termination** because a call to 'EventQueueInit' hooks some interrupts (the keyboard interrupt and the mouse driver callback function). If these interrupt are not reset to their initial values at program exit, the interrupt vectors will point into the already terminated program -- with disastrous results.

The 'EventQueueNextEvent' function copies the next event entry from the queue into the event buffer whose address is passed to it as an argument. It returns a non-zero value if these was an event available, zero otherwise.