Enqueue Add an element to the end of a queue

#include <OSUtils.h> **Operating System Utilities**

```
void         Enqueue(qEntry, theQueue );
QElemPtr     qEntry ;         address of a queue element
QHdrPtr      theQueue ;       address of a queue header
```

Enqueue adds an element to the end of a linked list known as a queue. The element itself (and not a copy) is hooked into the queue.

qEntry is the address of a variable-length QElem structure whose size and contents depend upon the type of queue. This packet contains a place for the queue linkage, the queue type, and the data of the queue element itself.

theQueue is the address of a 10-byte QHdr structure. This structure contains information about the queue - some type-specific flags and pointers to the first and last element in the queue.

Returns: none

Notes: The Macintosh Operating System uses queues to store and track such items as keyboard and mouse events, vertical retrace tasks, file I/O requests, and so forth.

Enqueue causes *theQueue* ->qTail and the previously-last queue element's qlink field to be updated to point to *qEntry*. The effect is that if you trace the queue links from start to finish, you will see all queue elements, including the newly-added element.

The **Enqueue** routine turns off interrupts for critical sections of its code. This makes it ideal for queue management for interrupt-driven programs which need to be concerned about simultaneous execution and deadlock.

For system-defined queues such as the event queue and vertical retrace task queue, *qEntry* must be one of the following predefined data types:

Struct Name	constant	value	Description
VBLTask	vType	1	vertical retrace task queue
ParamBlockRec	ioQType	2	file I/O or driver I/O queue
DrvQEl	drvQType	3	drive queue
EvQEl	evType	4	event queue
VCB	fsQType	5	volume control-block queue

For custom queues, the queue element structure must begin with 10 bytes - space for a "next element" pointer and a type code. Otherwise, the size and contents of the queue element are user-defined.

The following example uses **Enqueue** and **Dequeue** to manage a list of "bullets" in an arcade-style game. It initializes a queue header, adds some bullets to the queue, and repeatedly calls a routine that goes down the list, updating the screen positions of the bullets. When a bullet goes off-screen, it is removed from the queue.

Example

```
#include <OSUtils.h>
#define MAX_BULLETS 5

typedef struct BulletQEl {
    struct BulletQEl   *qLink;
    short       qType;              /* 0 for custom type */
    Rect        curLoc;             /* current location */
    short       xMove, yMove;       /* motion vectors */
    short       refCon;             /* holds array index */
} BulletQEl, *BulletQElPtr;

QHdr        bulletQHdr;                     /* the queue header */
BulletQEl   theBullets[MAX_BULLETS];        /* array of queue elements */

BulletTest()
{
    /* initialize the queue header */
    bulletQHdr.qFlags=0;                /* bits to find open array elements */
    bulletQHdr.qHead=0;
    bulletQHdr.qTail=0;

    AddBullet( 100,100, 5,3 );          /* enqueue some bullets */
    AddBullet( 100,110, 2,4 );
    AddBullet( 100,120, 20,0 );

    while ( 1 ) {                       /* assumes external exit */
        UpdateBullets();                /* track and draw each element */
    }
}
/* ------------------------------------------ */
AddBullet( x,y, xv,yv )
short x,y, xv,yv;
{
    short       j, availBullet;
    BulletQEl   *aBullet;

    availBullet=-1;
    for (j=0; j<MAX_BULLETS; j++ ) {
        if ( (bulletQHdr.qFlags & (1<<j)) == 0 ) {
            availBullet=j;
            break;
        }
    }
    if (availBullet == -1) return(1);          /* no openings */

    bulletQHdr.qFlags |= (1 << availBullet );  /* indicate slot is in use */
    aBullet = &theBullets[availBullet];
    SetRect( &aBullet->curLoc, x,y, x+5,y+5 ); /* initialize */
    aBullet->xMove=xv;
    aBullet->yMove=yv;
    aBullet->refCon=availBullet;
    Enqueue( aBullet, &bulletQHdr );           /* put in queue */
}
/* ------------------------------------------- */
UpdateBullets()
{
    short j;
    BulletQEl  *aBullet;

    aBullet = (BulletQElPtr)bulletQHdr.qHead;
    while ( aBullet != 0 ) {
        EraseRect( &aBullet->curLoc );
        OffsetRect(&aBullet->curLoc,aBullet->xMove,aBullet->yMove);
        if ( !PtInRect( topLeft(aBullet->curLoc),&thePort->portRect) ){
            Dequeue( aBullet, &bulletQHdr );        /* remove and mark it... */
            bulletQHdr.qFlags &= (~(1 << aBullet->refCon ) ); /* ...as open */
        }
        else {
            FillRect( &aBullet->curLoc, black );   /* draw at new location */
        }
        aBullet = aBullet->qLink;               /* get next element in queue */
    }
}
```