Event Loops

In applications that are event-driven (that is, which decide what to do at any time by receiving and responding to events), you can obtain information about events that are pending by calling **Event Manager** routines. Since you call these routines repeatedly, the section of code in which you request events from the **Event Manager** usually takes the form of a loop; this section of code is the event loop.

A simple event loop might look something like the one given in the following program. It consists of an endless loop that retrieves an event and decides whether it is a null event. If the event is not a null event, the event loop calls DoEvent, an application-defined procedure, to process the event. Otherwise, the procedure calls an application-defined idling procedure, DoIdle.

A simple event loop

```
// Assumes inclusion of <MacHeaders>

void EventLoop (void);
long GetSleep (void);
void AdjustCursor(Point where, RgnHandle cursorRgn);
void DoEvent (EventRecord *event);
void DoIdle (void);

void EventLoop ()
{
    RgnHandle cursorRgn;
    Boolean     gotEvent;
    EventRecord    event;

    cursorRgn = NewRgn();       // Pass empty region 1st time thru

    while (TRUE)  {   //  Loop forever
        gotEvent = WaitNextEvent(everyEvent, &event, GetSleep(),
                        cursorRgn);
        AdjustCursor(event.where, cursorRgn);
        if (gotEvent)
            DoEvent(&event);
        else
            DoIdle();
    }
}
```

The DoEvent procedure must determine what kind of event the call to **WaitNextEvent** retrieved and act accordingly. Notice that the parameter passed to DoEvent is the EventRecord received by **WaitNextEvent**. Essentially, the procedure is just a large conditional statement that branches according to the value of the *what* field of the EventRecord. The following program defines a simple DoEvent procedure.

Processing events

```
// Assumes inclusion of <MacHeaders>
#include <EPPC.h>

void DoEvent (EventRecord *event);
void DoMouseDown (EventRecord *event);
void DoMouseUp (EventRecord *event);
void DoKeyDown (EventRecord *event);
void DoActivate (EventRecord *event);
void DoUpdate (EventRecord *event);
void DoOSEvent (EventRecord *event);
void DoHighLevelEvent (EventRecord *event);

void DoEvent(EventRecord *event)
{
    switch (event->what )
    {
    case mouseDown :
        DoMouseDown(event);
        break;

    case mouseUp :
        DoMouseUp(event);
        break;

    case keyDown :
    case autoKey :
        DoKeyDown(event);
        break;

    case activateEvt :
        DoActivate(event);
        break;

    case updateEvt :
        DoUpdate(event);
        break;

    case osEvt :
        DoOSEvent(event);
        break;

    case kHighLevelEvent :
        DoHighLevelEvent(event);
        break;
    }
}   // DoEvent
```

The main addition to your application's event loop in System 7.0+ is the recognition of high-level events (using the constant kHighLevelEvent) and the appropriate processing of those events. The procedure defined here calls **DoHighLevelEvent,** an application-defined routine, to interpret the high-level event further.