**Note** If you actually look at the definition of an event record in the header files, you will find some other fields after `modifiers`. Technically, those are part of a `TaskMaster` record. We'll talk about those fields, and why they are in the event record, when we start using `TaskMaster`.

Let's stop and take a close look at just what the Event Manager is telling us. The first item in the event record is `what`. The `what` field tells us what kind of event has occurred. Here's a list of the various events the Event Manager can return.

| Name | Value | Description |
|---|---|---|
| `nullEvt` | 0 | Nothing happened. The mouse may have moved, but the mouse button hasn't been pressed, no key on the keyboard has been pressed, and the Event Manager didn't find any other sign that the person using the program is doing anything. |
| `mouseDownEvt` | 1 | The button on the mouse was pressed. This event occurs once, when the mouse button is originally pressed down. As long as the button is held down, you won't get another mouse down event. |
| `mouseUpEvt` | 2 | The button on the mouse was down, and has been released. |
| `keyDownEvt` | 3 | A key on the keyboard has been pressed. |
| `autoKeyEvt` | 5 | A key was pressed, and a `keyDownEvt` reported, but the key was held down. This event is reported periodically for as long as the key is held down. |
| `updateEvt` | 6 | A portion of a window needs to be redrawn (updated). |
| `activateEvt` | 8 | A window has been activated. |
| `switchEvt` | 9 | Switch events aren't used by the current system. Apple put them in just in case they ever wanted to generate a switch event, presumably for something like an application switcher or MultiFinder. |
| `deskAccEvt` | 10 | This event happens right before the CDA menu is brought up. |
| `driverEvt` | 11 | Device drivers can post these events. They really don't concern us. |
| `app1Evt` | 12 | You can define your own events. If you need one, this event code and the three that follow are for your use. |
| `app2Evt` | 13 | Application-defined event. |
| `app3Evt` | 14 | Application-defined event. |
| `app4Evt` | 15 | Application-defined event. |

Table 1-1: Event Manager Events

Some of these probably don't make much sense, yet. We'll look at several of these events in more detail as we learn more about the toolbox.

The `message` field tells more about the event that occurred. For example, the `message` field for a `keyDownEvt` contains the ASCII character code for the key that was pressed. The `message` field is different for each event, so we'll wait to look at this field in detail until we start looking at the individual events.

When you start the Event Manager, your Apple IIGS starts a clock that counts off each 1/60th of a second. Each time the clock increments, it adds one to the tick count. The tick count is recorded in the `when` field. You can use this value to tell when a particular event occurred, or more commonly, how much time elapsed between two events. For example, when a program looks for a double-click, the tick count is used to see if the two clicks on the mouse button were close enough together to be considered a double click. The `when` field is set even if no event occurred, so you can use it as a timer inside your event loop.

Like the `when` field, the `where` field is always set, even if the event is a null event. The `where` field is a point; it gives the position of the mouse in global coordinates. A point is a structure with two integer values, h and v. The value h is the horizontal position, starting from 0 and counting from the left of the screen. The vertical coordinate, v, counts from the top of the screen to the bottom. Global coordinates are the ones you would expect, with the 0,0 point at the top left of the screen. When we start to use windows, you'll learn about local coordinates. We'll take a closer look at global coordinates a little later in this lesson when we look at mouse events.

The `modifiers` field is a series of bit flags that give more information about the state of the computer. The `modifiers` field tells us if the mouse button is down, if the shift key is being pressed, and so forth. Figure 1-1 shows the modifier flags and what each bit is used for. It's worth taking a close look at figures like 1-1 – you find out some peculiar and interesting things about the toolbox when you do. For example, take a look at bits 6 and 7. Here we have two innocent bits that tell you when mouse button 0 and mouse button 1 are down, so you can tell which mouse button is being pressed. Well, that seems useful, but *my* mouse only has one button. Of course, Apple's desktop bus system that you use to connect your keyboard and mouse to the computer isn't limited to just the keyboard and mouse Apple sends with the computer. Apple's programmers put a lot of work into making the toolbox flexible enough to handle lots of options, many of which they don't intend to use themselves. That's a good design philosophy, but you need to keep alert for things like the two mouse buttons so you know what is available.

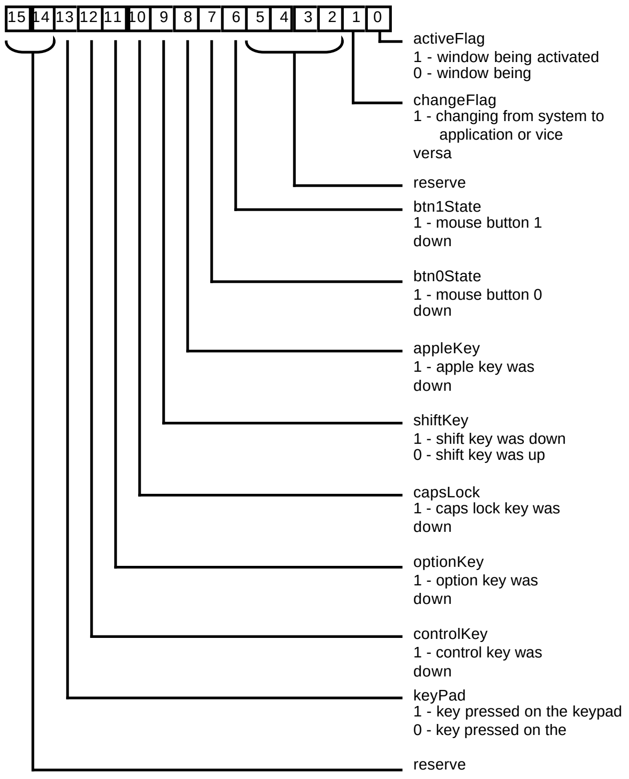

Figure 1-1: Modifier Flags

We've seen that `GetNextEvent` returns a value, which we have seen is `TRUE` if some event occurred, and `FALSE` if there was no event. We've seen how the event record is used to return lots of information about what event occurred – and some information, like the mouse position and modifiers, even when an event didn't occur. Looking back at `GetNextEvent`, though, there is one other parameter we haven't talked about. In some programs, you just may not need a particular event. The event mask parameter tells the Event Manager which events you want to see, and which events you want it to dump in the bit bucket. While this can be useful in some situations, it's really just as easy to ignore the events you don't want to handle in your event loop. For now, we'll use the value `everyEvent`, which tells the Event Manager to report each and every event it finds.

## Our First Executable Program

Well, we can finally create a complete desktop program. This one doesn't to much – it just brings up the desktop and waits for us to press a key, but hey, it's a start. Even though the program doesn't do much, we've covered a lot of ground, and it's important to see some of this new knowledge put to use.

There are a couple of things in this program you may not have expected; we'll talk about those in a moment. The event loop itself, though, is exactly what we've been leading up to in the last few pages.

```
/****************************************************************
*
* Keypress
*
* Start the desktop environment and wait for a keypress.
*
****************************************************************/

#pragma lint -1

#include <orca.h>
#include <Event.h>

BOOLEAN done;                           /* are we done, yet? */
EventRecord myEvent;                    /* event record */

/****************************************************************
*
* Main program
*
****************************************************************/

int main (void)

{
startdesk(640);
done = FALSE;
while (!done) {
   if (GetNextEvent(everyEvent, &myEvent))
      if (myEvent.what == keyDownEvt)
         done = TRUE;
   }
enddesk();
}
```

Listing 1-1: Wait for a Key Press