elcome Back!

Welcome to the second in what I hope will be a long series of columns crafted to help you learn to program on the Macintosh, or just learn to program better in general. Since this publication and its back issues will be forever available on the Apple Wizards website, I've decided to start from the beginning, and work up from there. What will you need to begin this journey? A basic introductory understanding of the C language, perhaps enough to create a 'hello world' program and run it. I'm going to assume that your only exposure to Mac OS programming was with a simple "console app" that you copied out of a book. In other words, you can pack light for this journey — there's plenty of food along the way.

You should also know what you can expect out of this. By the time the fourth installment of my column in AppleWizards hits the virtual stands, you can expect to know how to create a true Mac OS program, and you'll understand many of the basics. This doesn't mean you'll be ready to snap your fingers and create the next Tomb Raider. Mac OS programming in C is quite a lengthy and complex procedure. But it does create some of the smallest programs (in terms of size of finished applications), and is the best way to get a handle on just what your Mac is capable of. Also, by the fourth installment, you should be ready to start learning on your own quite easily. But more on that later.

Patience is a great virtue when it comes to programming. You'd be amazed at the kinds of trouble you'll encounter, and how easy it is to create really nasty bugs in your code. I suggest (and I do this myself) that you keep a diary of your problems. Not your progress, not what you've learned, but rather what you haven't learned. The reason for this is that you'll very quickly realize just what your weaknesses are. Also, once you get bitten again by the same exact out-of-place comma, semicolon, or bracket, you'll be ready to pull your few remaining hairs out. So keep a diary. Keep track of every problem you encounter, what the symptoms were, and most importantly, how you fixed it.

This column will focus on C and the underlying concepts which propel the Macintosh Operating System. However, C isn't the only way to go about programming. The right tool for the job is the key here. There are several excellent Rapid Application Design (RAD) tools on the market, such as RealBasic and SuperCard, which can do the hard and boring work for you. But to extend most of those languages, you'll need C. You'll never encounter boundaries you can't cross using C. Once you've written an entire application in C, you'll know everything you'll need to know about working with the Mac OS Toolbox, and if you don't, you'll know where to find it.

Lastly, there are many fine books on programming. I'll be doing short reviews of some of them in the coming months, so stay tuned. I've read around a dozen or so Mac Programming books, and although I can't say just which one was my favorite, or the easiest to understand, none of them really sucked. So if you're considering buying a Mac programming book, go right ahead.

Note
You'll hear a phrase thrown around a lot among Mac OS programmers, the "Toolbox." No, you don't need to go buy it. The easiest way to explain it is by example. Rather than let you try to figure out how the Mac speakers work, how to find the current "beep" sound, load it into RAM, create a sound channel, play it through the speakers, close the sound channel, dispose of the RAM, etc., Apple figured people would want to be able to do this regularly, so they created a function, "SysBeep," which when called will simply play the current beep sound. There are hundreds of such utilities, and they are collectively known as the Mac OS Toolbox. So now you know.

he Event Loop

The first concept I'm going to introduce you to will be the one of the most difficult and odd concepts you'll get the experience of wrapping your mind around in the entire course of programming on the Mac. Ever wonder how your Mac knows what to do when you click a menu, for instance? To your Mac, the world is nothing more than a series of events and responses. And that's the topic for today, the most important underlying concept in Macintosh programming. It's called the "Event Loop," and it is deceptively simple.

The event loop is like a train, and each program can be thought of as a station. Each app can tell the train to either drop off its cargo, continue to the next station, and more. In addition, each application can send events via the loop to other applications.

f you're experienced with console programming, a common paradigm in the academic world, then the concept of an event loop is often somewhat unnerving. Don't be afraid, the Mac environment is just the same as a normal console program, except that the command line is constantly being handed back and forth between various programs. Okay, it's a little eerie. Every program you have open is waiting, right now, for you to do something. And if you aren't doing anything, then all of those programs are doing something on their own — they're telling each other to wait until you do something else. We want your new program to fit in, so we'll teach it how to wait with the best of them. At the current time, there are only about eleven events that can occur on your Mac.

The Event List

- Activate Event
- Update Event
- Drive Event
- OS Event
- High-level Event
- Null Event
- Key-Down Event
- Key-Up Event
- AutoKey Event
- Mouse-Down Event
- Mouse-Up Event

he OS sends these events around, and it's going to be your program's job (and therefore yours) to figure out which ones to watch for, which ones to act on, and which ones to ignore. Luckily, the "event" contains information about itself in a special type of data structure called an Event Record. All Event Records have the same structure, they just have different contents. Here's the anatomy of the standard event record:

Event Record
• What
• Message
• Time
• Mouse Location
• Modifiers

The What section contains the type of event this is. It can be any of the eleven events mentioned above. For example, if your program is going to be AppleEvent aware (which is necessary for a program to be scriptable) then it's going to watch for High-level Events. If your program deals with files (most do), then you might want to watch for Drive Events. By checking the What section of the Event Record, you can decide how you want to respond.

The Message is where it gets interesting. In some cases, just knowing what happened is enough. For example, if you receive a mousedown event, you know that the mouse button was just pressed. But if you receive a key-down event, you probably want to know what key was pressed. The Message will sometimes be empty, but it will usually contain valuable data.

The Time is essentially a stamp that the Mac OS puts on each event so that you can know exactly when the event took place. It's actually the number of ticks since the Mac started up.

The Mouse Location (where) may be of great interest to you, especially if your app is watching for Mouse-Down events. You want to know where the mouse was when it was clicked, right?

And lastly the Modifier section will be flagged if there's one or more modifier keys being pressed. Modifier keys include the Command, Option, Control, Shift, and Caps Lock.

atience

Teaching your Mac to wait patiently is a simple thing. The Mac OS Toolbox function "WaitNextEvent" handles this for you. Simply call it, and your app will grab the next incoming event. This is the heart of the event loop. WaitNextEvent can take four parameters.

The first is the Event Mask. By using special keywords, you can tell your Mac to only watch for certain events. But most of the time, you'll want them all to at least stop on the way through, so you'll probably pass "everyEvent" in this parameter.

The second is a pointer to an Event Record. When WaitNextEvent finishes, this event record will be filled out with the complete event info.

The third is called a Sleep value, and it's the number of ticks that you're willing to go between checking for events. The larger the number, the more processing time you give other apps running at the same time. The lower the number gives your app more concurrent processing time. Hint: don't set it too low — that is rude.

The fourth parameter is one you'll probably not use often, as it's reserved for apps that change the appearance of the cursor. So for this one, we'll pass 'nil' as our parameter.

Because your Mac app is event-driven, you'll want to place the call to WaitNextEvent() inside a loop. And since there are eleven different results you could see from that function, you'll probably want to catch that result with a switch. Following is an example which brings all of this together. Notice that this is a function. Some Mac programmers just keep their event loop in Main(), but I'm a 'modular' programmer, and I really like functions. It's up to you, ultimately. Here's the code...

```
void   EventLoop( void ) // start the function


EventRecord
theEvent;   // declare variables

   while ( gDone == false ) // gDone is a global, so repeat
                            // the loop until we set
                            // gDone to true
```

```
        WaitNextEvent( everyEvent, &theEvent, 15L, nil );
        //                  \_________/   \________/   \_/   \_/
        //                         |               |          |      |

       // Capture all events               |          |      |
        //                                         |          |      |
        //      Pass a pointer to a record           |      |
        //                                                  |      |
        //          Check every 15 ticks (We use an      |
        //            'L' to tell it to force the          |
        //            15 to fill 4 bytes)                     |
        //                                                      |
        //             We don't change the cursor, so nil
        //

            switch ( theEvent.what ) // switch using What



case mouseDown:       // if it's a mousedown event...



HandleMouseDown( theEvent ); // handle that.




           break;                               // and exit the switch



case updateEvt:          // if it's an update event...



HandleMouseDown( theEvent ); // handle that.




           break;                               // and exit the switch

             // end switch
       // end while
 // end EventLoop function
```

ntil Next Time...

Hopefully you've gained some insight into how your Mac actually works, and how the Event Loop is the center of the Universe. This understanding is crucial to being a good Mac OS programmer.

In coming months, we'll skim some topics for a while and then dig into the code rather deeply.

Next month's column will be the bane of Systems everywhere, and the cause of more headaches and System Errors than nearly anything else… Memory.

Chilton Webb
chilton@applewizards.net