Carbon Event Timers

Author: Mark Szymczyk
Last Update: October 4, 2005

This article provides an introduction to Carbon event timers. After reading the article you will know how to write an event timer, install it, start it, and stop it.

Introduction

A timer is a function the operating system calls periodically. You get to specify how often the timer fires, which is how often the operating system calls the timer. Use a timer when you want to do something repeatedly in your program at regular time intervals. Examples of using timer in Mac OS X programs include the following:

- Maintaining a constant frame rate in games. Create an event timer that calls the game's main game loop and fires at the desired frame rate.
- Updating a clock display.
- Periodically opening a dialog box in shareware programs asking the user to register the program.
- Blinking a text insertion caret.

To use timers you need two variables in your program: one variable of type `EventLoopTimerRef`, and one variable of type `EventLoopTimerUPP`.

```
EventLoopTimerRef timer;
EventLoopTimerUPP timerUPP;
```

There are three tasks you must perform to use timers in your applications: install the timer, write the timer, and start the timer. I cover these tasks in the next three sections.

Installing a Timer

There are three tasks to perform to install a timer.

1) Get an event loop reference.
2) Create an event loop timer universal procedure pointer (UPP).
3) Call the function `InstallEventLoopTimer()`.

Most of you will be calling the function `GetMainEventLoop()` to get an event loop reference. As its name suggests, `GetMainEventLoop()` returns the main application thread's event loop. If you create a thread for your application and want to install an event timer for that thread, call the function `GetCurrentEventLoop()`, which returns the current thread's event loop.

```
EventLoopRef mainLoop;
mainLoop = GetMainEventLoop();
```

Universal procedure pointers (UPP) are a method of abstracting function pointers. What is important to know is that you have to create an event loop timer UPP to install a timer. Call the function `NewEventLoopTimerUPP()` to create the UPP. Supply your timer's function name to `NewEventLoopTimerUPP()`.

```
timerUPP = NewEventLoopTimerUPP(TimerFunction);
```

After creating the UPP you can install the timer by calling `InstallEventLoopTimer()`. This function takes six parameters. The first parameter is the event loop reference you got by calling `GetMainEventLoop()` or `GetCurrentEventLoop()`. The second parameter is the amount of time to wait before running the timer. If you want the timer to start firing immediately, pass the value `kEventDurationNoWait`. Passing the value `kEventDurationForever` tells the operating system to wait for you to tell it to fire the timer.

The third parameter is the timer's firing rate. If you set a firing rate of 0, the timer fires once. The fourth parameter is the UPP you created by calling `NewEventLoopTimerUPP()`. The fifth parameter is a pointer to data you're going to supply to the timer. Normally you supply a pointer to a class or data structure. The last parameter is the timer.

When you specify a firing rate or an amount of time to wait, Apple supplies constants that start with the prefix `kEventDuration`. You can specify firing rates and waiting times in nanoseconds,microseconds, milliseconds, seconds, minutes, hours, and days. The following example specifies a waiting time of 2 hours and a firing rate of 30 times per second:

```
InstallEventLoopTimer(mainLoop, (kEventDurationHour * 2),
            (kEventDurationSecond / 30),timerUPP,
            this, &timer);
```

If you want the timer to fire consistently, make sure the firing rate is slower than the time it takes for the timer to finish. If you set a firing rate of 60 times per second, but the timer takes one second to complete its tasks, the timer will end up firing approximately one time per second, not 60.

Writing a Timer

The timer function takes the form.

```
pascal void TimerFunction(EventLoopTimerRef theTimer,
    void* userData)
```

The `pascal` keyword tells the compiler to use Pascal language parameter passing conventions. Timer functions require the Pascal language conventions. The `userData` argument is the data you supplied when you called `InstallEventLoopTimer()`.

C++ programs that want timer functions to be member functions of a class must declare the timer to be a static function in the header file.

```
static pascal void TimerFunction(EventLoopTimerRef theTimer,
    void* userData);
```

You can give your timer function any name you want, but remember the name of the timer must match the name you supply to the function `NewEventTimerUPP()` when installing the timer.

What do you put in a timer? Generally you call a function from the timer. The following example calls the `GameLoop()` function for a `GameApp` class:

```
pascal void GameApp::GameLoopTimer(EventLoopTimerRef theTimer,
    void* userData)
{
    GameAppPtr currentApp = (GameAppPtr)userData;
    currentApp->GameLoop();
}
```

Starting and Stopping a Timer

Normally you install your program's timers when you're initializing your program. In this case you want to start the timers yourself instead of having the timers start firing immediately. To start playing a timer, call the function `SetEventLoopTimerNextFireTime()`.This function takes two arguments. The first argument is the timer. The second argument is the amount of time to wait to fire the timer. Pass the value `kEventDurationNoWait` as the second argument, which tells the operating system to start the timer.

```
OSStatus error;
error = SetEventLoopTimerNextFireTime(timer,
    kEventDurationNoWait);
```

Sometimes you need to temporarily turn off timers. If you use a timer to run your game's event loop, you want to turn off the timer when the player pauses the game. To pause the timer call `SetEventLoopTimerNextFireTime()`, but supply the value `kEventDurationForever` as the second argument. The following call pauses the timer:

```
OSStatus error;
error = SetEventLoopTimerNextFireTime(timer,
    kEventDurationForever);
```

Removing a Timer

When the user quits your program, you must remove the timer you installed as well as the UPP you created for the timer. Call the function `RemoveEventLoopTimer()` to remove the timer, supplying the timer name. Call the function `DisposeEventLoopTimerUPP()` to remove the UPP.

```
OSStatus error;
error = RemoveEventLoopTimer(timer);

DisposeEventLoopTimerUPP(timerUPP);
```

Conclusion

Unlike my previous articles, there is no source code to accompany this article. If you want to see examples of event timers, download the source code for my QuickTime and AGL articles. Both programs provide examples of using event timers in an application.