Timer Programming Topics

Data Management: Event Handling

2009-07-14

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction to Timers

Timers provide a way to perform delayed or periodic actions.

Organization of This Document

This topic describes how to use the `NSTimer` objects. The topic is divided into the following articles:

- "Timers" (page 7)
- "Using Timers" (page 9)

Timers

A timer provides a way to perform a delayed action or a periodic action. The timer waits until a certain time interval has elapsed and then fires, sending a specified message to a specified object. For example, you could create a timer that sends a message to a window, telling it to update itself after a certain time interval.

Timers and Run Loops

Timers are represented by `NSTimer` objects. They work in conjunction with `NSRunLoop` objects. `NSRunLoop` objects control loops that wait for input, and they use timers to help determine the maximum amount of time they should wait. When the timer's time limit has elapsed, the run loop fires the timer (causing its message to be sent), then checks for new input.

The run loop mode in which you register the timer must be running for the timer to fire. For applications built using the Application Kit, the `NSApplication` object runs the main thread's run loop for you. On secondary threads, however, you have to run the run loop yourself—see *Run Loops* for details.

Each run loop timer can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop.

Timing Accuracy

A timer is not a real-time mechanism; it fires only when one of the run loop modes to which the timer has been added is running and able to check if the timer's firing time has passed. Because of the various input sources a typical run loop manages, the effective resolution of the time interval for a timer is limited to on the order of 50-100 milliseconds. If a timer's firing time occurs while the run loop is in a mode that is not monitoring the timer or during a long callout, the timer does not fire until the next time the run loop checks the timer. Therefore, the actual time at which the timer fires potentially can be a significant period of time after the scheduled firing time.

A repeating timer reschedules itself based on the scheduled firing time, not the actual firing time. For example, if a timer is scheduled to fire at a particular time and every 5 seconds after that, the scheduled firing time will always fall on the original 5 second time intervals, even if the actual firing time gets delayed. If the firing time is delayed so far that it passes one or more of the scheduled firing times, the timer is fired only once for that time period; the timer is then rescheduled, after firing, for the next scheduled firing time in the future.

Alternatives to Timers

If you simply want to send a message at some point in the future, you can do so without using a timer. You can use `performSelector:withObject:afterDelay:` and related methods to invoke a method directly on another object. Some variants, such as `performSelectorOnMainThread:withObject:waitUntilDone:`, allow you to invoke the method on a different thread. You can also cancel a delayed message send using `cancelPreviousPerformRequestsWithTarget:` and related methods.

Using Timers

There are several aspects to using a timer. When you create a timer, you must configure it so that it knows what message to send to what object when it fires. You must then associate it with a run loop so that it will fire—some of the creation methods do this for you automatically. Finally, if you create a repeating timer, you must invalidate it when you want it to stop firing.

Creating and Scheduling a Timer

There are, broadly speaking, three ways to create a timer: scheduling a timer with the current run loop; creating a timer that you later register with a run loop; and initializing a timer with a given fire date. In all cases, you have to configure the timer to tell it what message it should send to what object when it fires, and whether it should repeat. With some methods, you may also provide a user info dictionary. You can put whatever you want into this dictionary that may be useful in the method that the timer invokes when it fires.

There are two ways to tell a timer what message it should send and the object to which it should send the message—by specifying each independently, or (in some cases) by using an instance of `NSInvocation`. If you specify the selector for the message directly, the name of the method does not matter but it must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

If you create an invocation object, you can specify whatever message you want. Note that an `NSTimer` object always instructs its `NSInvocation` object to retain its arguments, so you do not need to send `retainArguments` yourself. (For more about invocation objects, see Using NSInvocation in *Distributed Objects Programming Topics*.)

For the examples that follow, consider a timer controller object that declares methods to start and (in some cases) stop four timers configured in different ways. It has properties for two of the timers and a timer count, and three timer-related methods (`timerFireMethod:`, `invocationMethod:`, and `countedTimerFireMethod:`). It also provides a method to supply a user info dictionary.

```
@interface TimerController : NSObject {
    NSTimer *repeatingTimer;
    NSTimer *unregisteredTimer;
    NSUInteger timerCount;
}

@property (assign) NSTimer *repeatingTimer;
@property (retain) NSTimer *unregisteredTimer;
@property NSUInteger timerCount;

- (IBAction)startOneOffTimer:sender;

- (IBAction)startRepeatingTimer:sender;
- (IBAction)stopRepeatingTimer:sender;
```

```
- (IBAction)createUnregisteredTimer:sender;
- (IBAction)startUnregisteredTimer:sender;
- (IBAction)stopUnregisteredTimer:sender;

- (IBAction)startFireDateTimer:sender;

- (void)timerFireMethod:(NSTimer*)theTimer;
- (void)invocationMethod:(NSDate *)date;
- (void)countedTimerFireMethod:(NSTimer*)theTimer;

- (NSDictionary *)userInfo;

@end
```

The implementations of the user info method and two of the methods invoked by the timers might be as follows (`countedTimerFireMethod` is described in "Stopping a Timer" (page 12)):

```
- (NSDictionary *)userInfo {
    return [NSDictionary dictionaryWithObject:[NSDate date] forKey:@"StartDate"];
}

- (void)targetMethod:(NSTimer*)theTimer {

    NSDate *startDate = [[theTimer userInfo] objectForKey:@"StartDate"];
    NSLog(@"Timer started on %@", startDate);
}

- (void)invocationMethod:(NSDate *)date {

    NSLog(@"Invocation for timer started on %@", date);
}
```

Scheduled Timers

The following two class methods automatically register the new timer with the current `NSRunLoop` object in the default mode (`NSDefaultRunLoopMode`):

- `scheduledTimerWithTimeInterval:invocation:repeats:`
- `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:`

The following example shows how you can schedule a one-off timer that uses a selector:

```
- (IBAction)startOneOffTimer:sender {

    [NSTimer scheduledTimerWithTimeInterval:2.0
             target:self
             selector:@selector(targetMethod:)
             userInfo:[self userInfo]
             repeats:NO];
}
```

The timer is automatically fired by the run loop after 2 seconds, and is then removed from the run loop.

The next example shows how you can schedule a repeating timer, that again uses a selector:

```
- (IBAction)startRepeatingTimer:sender {

    NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:0.5
                            target:self selector:@selector(timerFireMethod:)
                            userInfo:[self userInfo] repeats:YES];
    self.repeatingTimer = timer;
}
```

If you create a repeating timer, you often need to save a reference to it so that you can stop the timer at a later stage (see "Initializing a Timer with a Fire Date" (page 11) for an example of when this is not the case).

Unscheduled Timers

The following methods create timers that you may schedule at a later time by sending the message `addTimer:forMode:` to an `NSRunLoop` object.

- `timerWithTimeInterval:invocation:repeats:`
- `timerWithTimeInterval:target:selector:userInfo:repeats:`

The following example shows how you can create a timer that uses an invocation object in one method, and then, in another method, start the timer by adding it to a run loop:

```
- (IBAction)createUnregisteredTimer:sender {

    NSMethodSignature *methodSignature = [self
methodSignatureForSelector:@selector(invocationMethod:)];
    NSInvocation *invocation = [NSInvocation
invocationWithMethodSignature:methodSignature];
    [invocation setTarget:self];
    [invocation setSelector:@selector(invocationMethod:)];
    NSDate *startDate = [NSDate date];
    [invocation setArgument:&startDate atIndex:2];

    NSTimer *timer = [NSTimer timerWithTimeInterval:0.5 invocation:invocation
repeats:YES];
    self.unregisteredTimer = timer;
}

- (IBAction)startUnregisteredTimer:sender {
    if (unregisteredTimer != nil) {
        NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
        [runLoop addTimer:unregisteredTimer forMode:NSDefaultRunLoopMode];
    }
}
```

Initializing a Timer with a Fire Date

You can allocate an `NSTimer` object yourself and send it an `initWithFireDate:interval:target:selector:userInfo:repeats:` message. This allows you to specify an initial fire date independently of the repeat interval. Once you've created a timer, the only property you can modify is its firing date (using `setFireDate:`). All other parameters are immutable after creating the timer. To cause the timer to start firing, you must add it to a run loop.

The following example shows how you can create a timer with a given start time (in this case, one second in the future), and then start the timer by adding it to a run loop:

```
- (IBAction)startFireDateTimer:sender {
    NSDate *fireDate = [NSDate dateWithTimeIntervalSinceNow:1.0];
    NSTimer *timer = [[NSTimer alloc] initWithFireDate:fireDate
                                      interval:0.5
                                      target:self
                                      selector:@selector(countedtargetMethod:)
                                      userInfo:[self userInfo]
                                      repeats:YES];

    timerCount = 1;
    NSRunLoop *runLoop = [NSRunLoop currentRunLoop];
    [runLoop addTimer:timer forMode:NSDefaultRunLoopMode];
    [timer release];
}
```

In this example, although the timer is configured to repeat, it will be stopped after it has fired three times by the `countedtargetMethod:` that it invokes—see "Stopping a Timer" (page 12).

Stopping a Timer

If you create a non-repeating timer, there is no need to take any further action. It automatically stops itself after it fires. For example, there is no need to stop the timer created in the "Initializing a Timer with a Fire Date" (page 11). If you create a repeating timer, however, you stop it by sending it an `invalidate` message. You can also send a non-repeating timer an `invalidate` message before it fires to prevent it from firing.

The following examples show the stop methods for the timers created in the previous examples:

```
- (IBAction)stopRepeatingTimer:sender {
    [repeatingTimer invalidate];
    self.repeatingTimer = nil;
}

- (IBAction)stopUnregisteredTimer:sender {
    [unregisteredTimer invalidate];
    self.unregisteredTimer = nil;
}
```

You can also invalidate a timer from the method it invokes. For example, the method invoked by the timer shown in "Initializing a Timer with a Fire Date" (page 11) might look like this:

```
- (void)countedtargetMethod:(NSTimer*)theTimer {

    NSDate *startDate = [[theTimer userInfo] objectForKey:@"StartDate"];
    NSLog(@"Timer started on %@; fire count %d", startDate, timerCount);

    timerCount++;
    if (timerCount > 3) {
        [theTimer invalidate];
    }
}
```

This will invalidate the timer after it has fired three times. Since the timer is passed as an argument to the method it invokes, there may be no need to maintain the timer as a variable. Typically, however, you might nevertheless keep a reference to the timer in case you want the option of stopping it earlier.

Memory Management

Because the run loop maintains the timer, from the perspective of memory management there's typically no *need* to keep a reference to a timer *after you've scheduled it*. Since the timer is passed as an argument when you specify its method as a selector, you can invalidate a repeating timer when appropriate within that method. In many situations, however, you also want the option of invalidating the timer—perhaps even before it starts. In this case, you *do* need to keep a reference to the timer, so that you can send it an `invalidate` message whenever appropriate. If you create an unscheduled timer (see "Unscheduled Timers" (page 11)), then you must maintain a strong reference to the timer (in a reference-counted environment, you retain it) so that it is not deallocated before you use it.

In keeping with standard Cocoa memory management rules, a timer maintains a strong reference to its user info dictionary (that is, in a reference-counted environment a timer retains it target, and in a garbage-collected environment it has a strong reference to the target.). There is no need to maintain the contents of the dictionary elsewhere. Perhaps more importantly, a timer also maintains a strong reference to its target. This means that as long as a timer remains valid (and you otherwise properly abide by memory management rules), its target will not be deallocated. As a corollary, this means that it does not make sense for a timer's target to try to invalidate the timer in its `dealloc` or `finalize` method—neither method will be invoked as long as the timer is valid.

Document Revision History

This table describes the changes to *Timer Programming Topics*.

Date	Notes
2009-07-14	Corrected minor typographical errors.
2008-11-19	Corrected typographical errors.
2008-10-15	Updated to discuss use of timers in a garbage-collected environment.
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.