Class

Timer

A timer that fires after a certain time interval has elapsed, sending a specified message to a target object.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class Timer
```

Overview

Timers work in conjunction with run loops. Run loops maintain strong references to their timers, so you don't have to maintain your own strong reference to a timer after you have added it to a run loop.

To use a timer effectively, you should be aware of how run loops operate. See Threading Programming Guide for more information.

A timer is not a real-time mechanism. If a timer's firing time occurs during a long run loop callout or while the run loop is in a mode that isn't monitoring the timer, the timer doesn't fire until the next time the run loop checks the timer. Therefore, the actual time at which a timer fires can be significantly later. See also Timer Tolerance.

`Timer` is toll-free bridged with its Core Foundation counterpart, `CFRunLoopTimer`. See Toll-Free Bridging for more information.

Comparing Repeating and Nonrepeating Timers

You specify whether a timer is repeating or nonrepeating at creation time. A nonrepeating timer fires once and then invalidates itself automatically, thereby preventing the timer from firing again. By contrast, a repeating timer fires and then reschedules itself on the same run loop. A repeating timer always schedules itself based on the scheduled firing time, as opposed to 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.

Timer Tolerance

In iOS 7 and later and macOS 10.9 and later, you can specify a tolerance for a timer (`tolerance`). This flexibility in when a timer fires improves the system's ability to optimize for increased power savings and responsiveness. The timer may fire at any time between its scheduled fire date and the scheduled fire date plus the tolerance. The timer doesn't fire before the scheduled fire date. For repeating timers, the next fire date is calculated from the original fire date regardless of tolerance applied at individual fire times, to avoid drift. The default value is zero, which means no additional tolerance is applied. The system reserves the right to apply a small amount of tolerance to certain timers regardless of the value of the `tolerance` property.

As the user of the timer, you can determine the appropriate tolerance for a timer. A general rule, set the tolerance to at least 10% of the interval, for a repeating timer. Even a small amount of tolerance has significant positive impact on the power usage of your application. The system may enforce a maximum value for the tolerance.

Scheduling Timers in Run Loops

You can register a timer in only one run loop at a time, although it can be added to multiple run loop modes within that run loop. There are three ways to create a timer:

- Use the `scheduledTimer(timeInterval:invocation:repeats:)` or `scheduledTimer(timeInterval:target:selector:userInfo:repeats:)` class method to create the timer and schedule it on the current run loop in the default mode.
- Use the `init(timeInterval:invocation:repeats:)` or `init(timeInterval:target:selector:userInfo:repeats:)` class method to create the timer object without scheduling it on a run loop. (After creating it, you must add the timer to a run loop manually by calling the `add(_:forMode:)` method of the corresponding `RunLoop` object.)
- Allocate the timer and initialize it using the `init(fireAt:interval:target:selector:userInfo:repeats:)` method. (After creating it, you must add the timer to a run loop manually by calling the `add(_:forMode:)` method of the corresponding `RunLoop` object.)

Once scheduled on a run loop, the timer fires at the specified interval until it is invalidated. A nonrepeating timer invalidates itself immediately after it fires. However, for a repeating timer, you must invalidate the timer object yourself by calling its `invalidate()` method. Calling this method requests the removal of the timer from the current run loop; as a result, you should always

call the `invalidate()` method from the same thread on which the timer was installed. Invalidating the timer immediately disables it so that it no longer affects the run loop. The run loop then removes the timer (and the strong reference it had to the timer), either just before the `invalidate()` method returns or at some later point. Once invalidated, timer objects cannot be reused.

After a repeating timer fires, it schedules the next firing for the nearest future date that is an integer multiple of the timer interval after the last scheduled fire date, within the specified `tolerance`. If the time taken to call out to perform a selector or invocation is longer than the specified interval, the timer schedules only the next firing; that is, the timer doesn't attempt to compensate for any missed firings that would have occurred while calling the specified selector or invocation.

Subclassing Notes

Do not subclass `Timer`.

Topics

Creating a Timer

```
class func scheduledTimer(withTimeInterval: TimeInterval, repeats: Bool, block: (Timer) -> Void) -> Timer
```

Creates a timer and schedules it on the current run loop in the default mode.

```
class func scheduledTimer(timeInterval: TimeInterval, target: Any, selector: Selector, userInfo: Any?, repeats: Bool) -> Timer
```

Creates a timer and schedules it on the current run loop in the default mode.

```
class func scheduledTimer(timeInterval: TimeInterval, invocation: NSInvocation, repeats: Bool) -> Timer
```

Creates a new timer and schedules it on the current run loop in the default mode.

```
init(timeInterval: TimeInterval, repeats: Bool, block: (Timer) -> Void)
```

Initializes a timer object with the specified time interval and block.

```
init(timeInterval: TimeInterval, target: Any, selector: Selector, userInfo: Any?, repeats: Bool)
```

Initializes a timer object with the specified object and selector.

`init(timeInterval: TimeInterval, invocation: NSInvocation, repeats: Bool)`

Initializes a timer object with the specified invocation object.

`convenience init(fire: Date, interval: TimeInterval, repeats: Bool, block: (Timer) -> Void)`

Initializes a timer for the specified date and time interval with the specified block.

`init(fireAt: Date, interval: TimeInterval, target: Any, selector: Selector, userInfo: Any?, repeats: Bool)`

Initializes a timer using the specified object and selector.

Firing a Timer

`func fire()`

Causes the timer's message to be sent to its target.

Stopping a Timer

`func invalidate()`

Stops the timer from ever firing again and requests its removal from its run loop.

Retrieving Timer Information

`var isValid: Bool`

A Boolean value that indicates whether the timer is currently valid.

`var fireDate: Date`

The date at which the timer will fire.

`var timeInterval: TimeInterval`

The timer's time interval, in seconds.

`var userInfo: Any?`

The receiver's `userInfo` object.

Configuring Firing Tolerance

`var tolerance: TimeInterval`

The amount of time after the scheduled fire date that the timer may fire.

Firing Messages as a Combine Publisher

```
static func publish(every: TimeInterval, tolerance: TimeInterval?, on: RunLoop, in: RunLoop.Mode, options: RunLoop.SchedulerOptions?) -> Timer.TimerPublisher
```

Returns a publisher that repeatedly emits the current date on the given interval.

```
class TimerPublisher
```

A publisher that repeatedly emits the current date on a given interval.

Relationships

Inherits From

NSObject

Conforms To

CVarArg
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSObjectProtocol

See Also

Run Loop Scheduling

```
class RunLoop
```

The programmatic interface to objects that manage input sources.