Chapter 9

TIMERS

CHAPTER 9 TIMERS

Goal

To understand and use NSTimer for scheduling automatic, repeating message invocations.

Prerequisites

Understanding of target/action and message-based programming.

Objectives

At the end of this section, you will be able to:

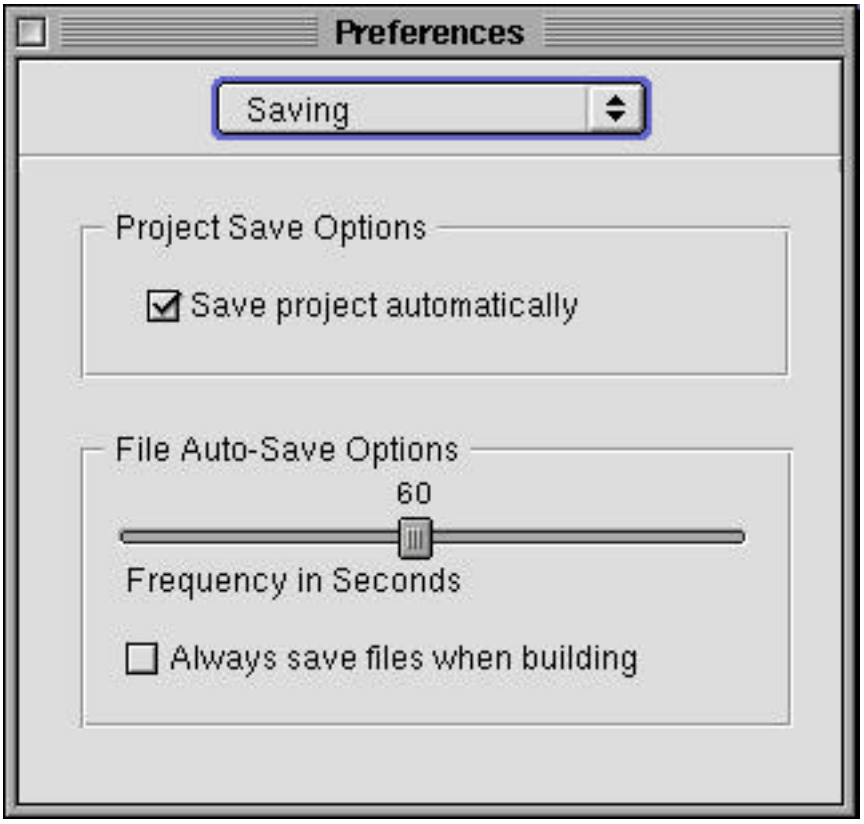
- Schedule an NSTimer instance to repeatedly message a custom object at a given frequency
- Stop an NSTimer when no longer desired
- Identify other classes and features involved with tracking time

Reading

NSTimer class reference in the Foundation

NSDate class reference in the Foundation

NSCalendarDate class reference in the Foundation



Applications often schedule timed events

Applications frequently have two related needs:

» To automate certain functions

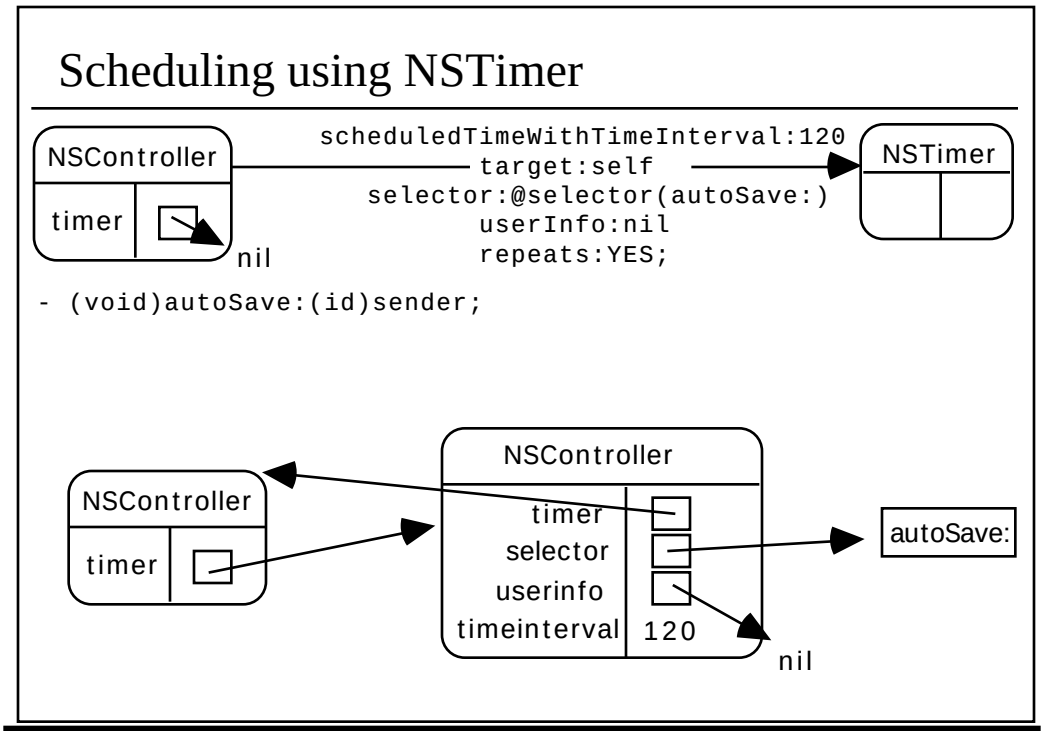
» To schedule automatic events at specific times

A great example of a scheduled event is the basic automatic backup features provided by most file-based applications.

This poses two questions:

» How do I schedule an event at a given time?

» What can I do with time objects?



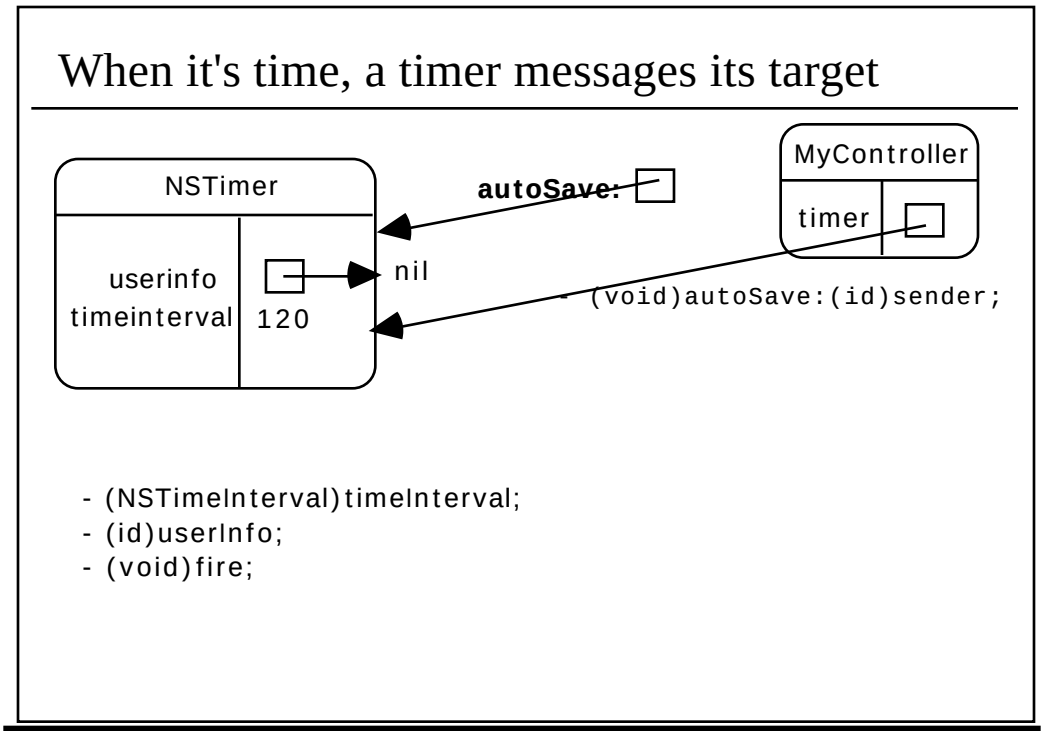
Scheduling using NSTimer

NSTimer is a Foundation object for setting up an alarm, repeating or not. You ask the factory for an instance which is immutable and preconfigured with your parameters. You will probably want to **retain** the timer and assign it to an outlet for future reference.

NSTimer takes a **target** and a **selector**. Think of it much like scheduling a target/action control to be activated at some known time in the future.

The unit for the time interval is seconds—it is a double and can include fractions. The boolean parameter **repeats:** indicates whether or not the alarm should be automatically rescheduled after the current one is fired.

Parameter passing to the pending target/selector message is more generalized than in the target/action interface. You can pass an arbitrary dictionary of parameters with the **userInfo** parameter. Since the message is typically just for activating a timed method, **userInfo** is often nil.



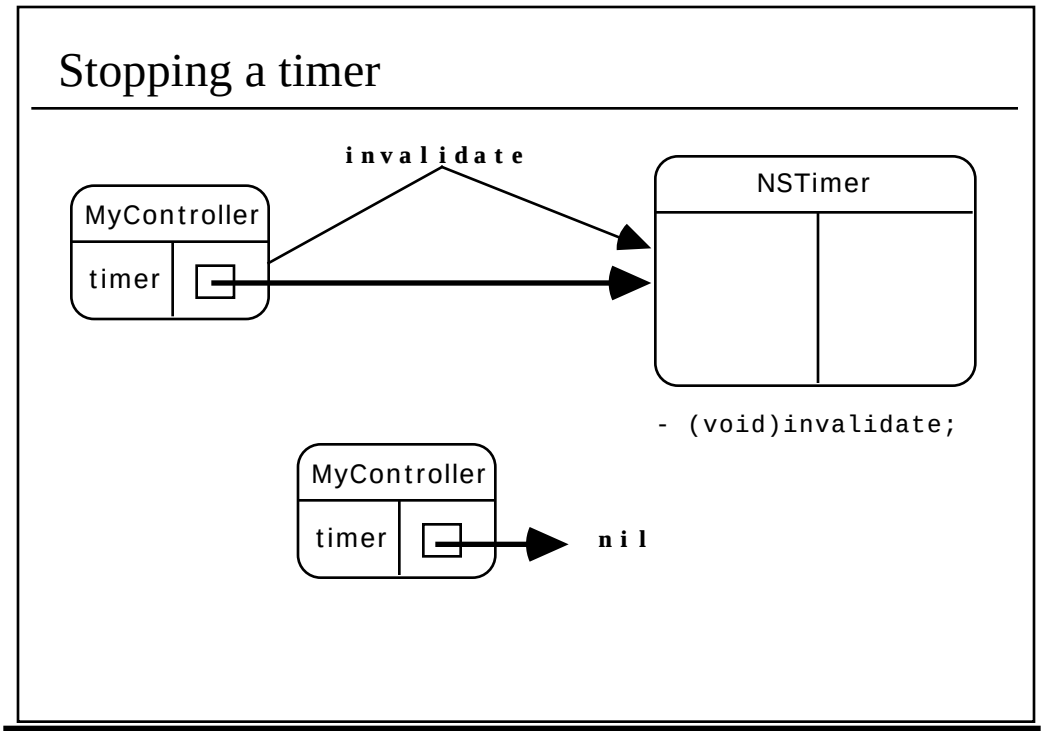
When it's time, a timer messages its target

When the time arrives, a timer event is placed in the event queue and a message, the stored selector, is sent to the designated target object. Your application begins automatically backing up the open and modified documents.

The responsible NSTimer instance passes itself as the **sender** reinforcing its similarity to the simple mechanics of target/action. You may query it to check the scheduled time interval, or retrieve the userInfo parameter dictionary.

If necessary, at other predetermined points in the life of your application, you can directly fire the NSTimer with the **fire** message.

Now how do you stop the timer?



Stopping a timer

Whether within a method called by the timer directly or in any context that provides access to the active timer, you stop a timer by sending it the **invalidate** message. Any pending event for that particular timer instance is ignored and all future scheduling is stopped. The NSTimer instance itself should be considered invalid and henceforth ignored. It will be released for you. Do not do it yourself.

Your own cleanup involves setting your timer outlet to **nil** to indicate that there is no scheduled timer and that you should instantiate and reassign when it's time to reschedule the alarm.

NSInvocation—objectifying arbitrary methods calls

NSInvocation is another Foundation object that defines a complete Objective-C message statement as an object instance with attributes:

- **target**
- **selector**
- **methodSignature**
- **arguments**

An NSInvocation instance can be fired directly with invoke, scheduled with a timer, or passed to another local or remote object that needs to use it.

Once it is called, an NSInvocation provides the return value completing a two-way communication.

NSCalendarDate - working with dates and times

Unit
second

Capabilities
current date
calculating and comparing dates
extracting date fields, from year to second
formatting (programmatic and NSDateFormatter)
localized

Related types and classes
NSTimeInterval (double)
NSTimeZone

NSCalendarDate—working with dates and times

NSCalendarDate is a Foundation object for dealing with time. You can ask the NSCalendarDate factory for the current time, format it and display it. Formatting is controlled with strings including literals and formatting specifiers much in the style of **printf**. Given any NSCalendarDate instance, which represents some arbitrary time. You can extract interesting fields from it such as the year, the hour, the second. And you can easily play time games for scheduling purposes like asking for a time object that represents the date 42 seconds from now or 2 weeks ago.

Any time object can identify itself in terms of an NSTimeInterval from right now which is an ideal parameter for scheduling a timer.

For display purposes, you can request locale-specific formatted strings so that time display is inherently internationalized. Further global accommodations are easily handled for display or calculations through NSTimeZone objects.

Important ideas from this section

» Use NSTimer for scheduling one-time or repeating message invocations to automate aspects of your application

» NSTimer is an immutable object that should be forgotten once invalidated

» NSCalendarDate provides an object-oriented interface to date and time handling with rich behavior for both custom display and internal calculations

Classes featured in this section

» NSTimer

» NSInvocation

» NSCalendarDate

» NSTimeInterval (data type)

» NSTimeZone

REVIEW TIMERS

1. Basic use of the NSTimer class usually involves three distinct steps. List them.

2. Assume your application may schedule up to 10 different timers concurrently. How would you distinguish one from the other? How can you tell which one sent the current message?

3. Timers are not always precise. Although they never fire early, they may often fire late. Can you say why this is the case?