
Date and Time Programming Guide for Cocoa

Data Management: Dates, Times, & Numbers



2009-07-21



Apple Inc.
© 2002, 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Cocoa, Mac, Mac OS, Monaco, and Objective-C are trademarks of Apple Inc., registered in the United States and other countries.

iPhone and Numbers are trademarks of Apple Inc.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to Date and Time Programming Guide for Cocoa 7

Organization of This Document 7
See Also 7

Dates 9

About Dates 9
Creating Date Objects 9
Basic Date Calculations 10

Calendars, Date Components, and Calendar Units 11

Calendar Basics 11
Date Components and Calendar Units 11
Creating a Date from Components 12

Calendrical Calculations 15

Adding Components to a Date 15
Temporal Differences 16
Converting from One Calendar to Another 17

Using Time Zones 19

Creating Time Zones 19
Application Default Time Zone 19

Legacy API: NSDate 21

Using NSDate 21
String Representations of NSDate Objects 22
The Calendar Format 22
String Representations of NSDate Objects 24

Document Revision History 25

Listings

Dates 9

- Listing 1 Creating dates with time intervals 10
- Listing 2 Creating dates by adding a time interval 10

Calendars, Date Components, and Calendar Units 11

- Listing 1 Creating calendar objects 11
- Listing 2 Creating a date components object 12
- Listing 3 Getting a date's components 12
- Listing 4 Creating a date from components 12

Calendrical Calculations 15

- Listing 1 An hour and a half from now 15
- Listing 2 Getting the Sunday in the current week 15
- Listing 3 Getting the beginning of the week 16
- Listing 4 Getting the difference between two dates 16
- Listing 5 Converting date components from one calendar to another 17

Introduction to Date and Time Programming Guide for Cocoa

This document describes how to handle date and time data using Cocoa.

You should read this document if your application keeps track of dates and times, particularly if it compares and manipulates elapsed time and points in time, such as calendar dates.

Organization of This Document

This document contains three articles and one appendix:

- [“Dates”](#) (page 9) describes the `NSDate` class, which represents dates and times in Cocoa. It explains how to create date objects and how to perform date-based calculations.
- [“Calendars, Date Components, and Calendar Units”](#) (page 11) describes the basic features of the `NSCalendar` class.
- [“Calendrical Calculations”](#) (page 15) describes the basic features of the `NSDateComponents` class and explains how to create calendars and perform calendrical computations, including getting the component elements of dates.
- [“Using Time Zones”](#) (page 19) explains the behavior of time zone objects defined by the `NSTimeZone` class.
- [“Legacy API: NSCalendarDate”](#) (page 21) provides information about the class that represents a Gregorian calendar, `NSCalendarDate`. This class does not exist in iPhone OS and is deprecated in Mac OS X. All of the capabilities of `NSCalendarDate` are duplicated by other, recommended classes, as described in the other articles of this document.

See Also

Data Formatting Programming Guide for Cocoa explains how to create and format user-readable strings from date objects, and how to create date objects from formatted strings.

Dates

This article describes the class you use to represent dates and times in Cocoa.

About Dates

Cocoa represents dates and times as date objects, instances of the `NSDate` class. `NSDate` is one of the fundamental Cocoa value objects. A date object represents an invariant point in time. Because a date is a point in time, it implies clock time as well as a day, so there is no way to define a date object to represent a day without a time.

To understand how Cocoa handles dates, you must consider `NSCalendar` and `NSDateComponents` objects as well. In a nontechnical context, a point in time is usually represented by a combination of clock time and a day on a particular calendar (such as a Gregorian or Hebrew calendar—supporting different calendars is important for localization). Likewise in Cocoa, you use a particular type of calendar object to decompose a date object into its date components such as year, month, day, hour, and minute. Conversely, you can use a calendar to create a date object from date components. Calendar and date component objects are described in more detail in [“Calendars, Date Components, and Calendar Units”](#) (page 11).

`NSDate` provides methods for creating dates, comparing dates, and computing intervals. Date objects, as an invariant point in time, are immutable. The standard unit of time for date objects is a value typed as a floating point value types as `NSTimeInterval` and is expressed as seconds. This type makes possible a wide and fine-grained range of date and time values, giving precision within milliseconds for dates 10,000 years apart.

`NSDate` computes time as seconds relative to the absolute reference time: the first instant of 1 January 2001, Greenwich Mean Time (GMT). Dates before then are stored as negative numbers; dates after then are stored as positive numbers. The sole primitive method of `NSDate`, `timeIntervalSinceReferenceDate`, provides the basis for all the other methods in the `NSDate` interface. `NSDate` converts all date and time representations to and from `NSTimeInterval` values that are relative to the absolute reference date.

Cocoa implements time according to the Network Time Protocol (NTP) standard, which is based on Coordinated Universal Time.

Creating Date Objects

If you want a date that represents the current time, you allocate an `NSDate` object and initialize it with `init`:

```
NSDate *now = [[NSDate alloc] init];
```

or use the `NSDate` class method `date` to create the date object. If you want some time other than the current time, you can use one of `NSDate`'s `initWithTimeInterval...` or `dateWithTimeInterval...` methods; typically, however, you use a more sophisticated approach employing a calendar and date components as described in [“Calendar Basics”](#) (page 11).

The `initWithTimeInterval...` methods initialize date objects relative to a particular time, which the method name describes. You specify (in seconds) how much more recent or how much more in the past you want your date object to be. To specify a date that occurs earlier than the method's reference date, use a negative number of seconds.

Listing 1 defines two date objects. The `tomorrow` object is exactly 24 hours from the current date and time, and `yesterday` is exactly 24 hours earlier than the current date and time.

Listing 1 Creating dates with time intervals

```
NSTimeInterval secondsPerDay = 24 * 60 * 60;
NSDate *tomorrow = [[NSDate alloc]
    initWithTimeIntervalSinceNow:secondsPerDay];
NSDate *yesterday = [[NSDate alloc]
    initWithTimeIntervalSinceNow:-secondsPerDay];
```

Listing 2 shows how to get new date objects with date-and-time values adjusted from existing date objects using `addTimeInterval:`. (Notice that `addTimeInterval:` returns a new date object—it does not modify the existing date object.)

Listing 2 Creating dates by adding a time interval

```
NSTimeInterval secondsPerDay = 24 * 60 * 60;
NSDate *today = [[NSDate alloc] init];
NSDate *tomorrow, *yesterday;

tomorrow = [today addTimeInterval:secondsPerDay];
yesterday = [today addTimeInterval:-secondsPerDay];
```

Basic Date Calculations

To compare dates, use the `isEqualToDate:`, `compare:`, `laterDate:`, and `earlierDate:` methods. These methods perform exact comparisons, which means they will detect subsecond differences between dates. You might want to compare dates with a less fine granularity. For example, you might want to consider two dates equal if they are within a minute of each other. If this is the case, use `timeIntervalSinceDate:` to compare the two dates. The following code fragment shows how to use `timeIntervalSinceDate:` to see if two dates are within one minute (60 seconds) of each other.

```
if (fabs([date2 timeIntervalSinceDate:date1]) < 60) ...
```

To obtain the difference between a date object and another point in time, send a `timeIntervalSince...` message to the date object. For example, `timeIntervalSinceNow` gives you the time, in seconds, between the current time and the receiving date object.

To get the component elements of a date, such as the day of the week, use an `NSDateComponents` object in conjunction with an `NSCalendar` object. This technique is described in “[Calendar Basics](#)” (page 11).

Calendars, Date Components, and Calendar Units

Calendars encapsulate information about systems of reckoning time in which the beginning, length, and divisions of a year are defined. They provide information about the calendar and support for calendrical computations such as determining the range of a given calendrical unit and adding units to a given absolute time. This article describes the basic features of the `NSCalendar` class, and the `NSDateComponents` class, instances of which describe the component elements of a date required for calendrical computations.

Calendar Basics

In Mac OS X v10.4 and later and iPhone OS 2.0 and later, the `NSCalendar` class provides an implementation of calendars for Cocoa. Cocoa provides data for several different calendars, including Buddhist, Gregorian, Hebrew, Islamic, and Japanese (which calendars are supported depends on the release of the operating system—you should check the `NSLocale` class to determine which are supported on a given release). `NSCalendar` is closely associated with the `NSDateComponents` class, instances of which describe the component elements of a date required for calendrical computations.

When you create a calendar object, you specify an identifier for the calendar you want. Calendars are specified by constants in `NSLocale`. You can get the calendar for the user's preferred locale most easily using the `NSCalendar` method `currentCalendar`; you can get the default calendar from any `NSLocale` object using the key `NSLocaleCalendar`. Listing 1 shows how to create a calendar object for the Japanese calendar and for the current user.

Listing 1 Creating calendar objects

```
NSCalendar *currentCalendar = [NSCalendar currentCalendar];

NSCalendar *japaneseCalendar = [[NSCalendar alloc]
                               initWithCalendarIdentifier:NSJapaneseCalendar];

NSCalendar *usersCalendar = [[NSCalendar alloc] initWithCalendarIdentifier:
                             [[NSLocale currentLocale] objectForKey:NSLocaleCalendar]];
```

Here, `usersCalendar` and `currentCalendar` are equal, although they are different objects.

Date Components and Calendar Units

You represent the component elements of a date—such as the year, day, and hour—using an instance of `NSDateComponents`. *An instance of `NSDateComponents` is meaningless in itself*; you need to know what calendar it is interpreted against, and you need to know whether the values are absolute values of the units or quantities of the units (see [“Adding Components to a Date”](#) (page 15) for an example of using `NSDateComponents` to specify quantities of units).

In a calendar, day, week, weekday, month, and year numbers are generally 1-based, but there may be calendar-specific exceptions. Ordinal numbers, where they occur, are 1-based. Some calendars may have to map their basic unit concepts into the year/month/week/day/... nomenclature. The particular values of the unit are defined by each calendar and are not necessarily consistent with values for that unit in another calendar.

[Listing 2](#) (page 12) shows how you can create a date components object that you could use either to create the date where the year unit is 2004, the month unit is 5, and the day unit is 6 (in the Gregorian calendar this would be May 6th, 2004), or to add 2004 year units, 5 month units, and 6 day units to an existing date. The value of `weekday` is undefined since it is not otherwise specified.

Listing 2 Creating a date components object

```
NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay:6];
[components setMonth:5];
[components setYear:2004];

NSInteger weekday = [components weekday]; // Undefined (==
NSUndefinedDateComponent)
```

To decompose a date into constituent components, you use the `NSCalendar` method `components:fromDate:`. In addition to the date itself, you need to specify the components to be returned in the `NSDateComponents` object. For this, the method takes a bit mask composed of `NSCalendarUnit` constants. There is no need to specify any more components than those in which you are interested. [Listing 3](#) shows how to calculate today's day and weekday.

Listing 3 Getting a date's components

```
NSDate *today = [NSDate date];
NSCalendar *gregorian = [[NSCalendar alloc]
    initWithCalendarIdentifier:NSGregorianCalendar];
NSDateComponents *weekdayComponents =
    [gregorian components:(NSDayCalendarUnit |
NSWeekdayCalendarUnit) fromDate:today];
NSInteger day = [weekdayComponents day];
NSInteger weekday = [weekdayComponents weekday];
```

Creating a Date from Components

You can configure an instance of `NSDateComponents` to specify the components of a date and then use the `NSCalendar` method `dateFromComponents:` to create the corresponding date object. You can provide as many components as you need (or choose to). When there is incomplete information to compute an absolute time, default values such as 0 and 1 are usually chosen by a calendar, but this is a calendar-specific choice. If you provide inconsistent information, calendar-specific disambiguation is performed (which may involve ignoring one or more of the parameters).

[Listing 4](#) shows how to create a date object to represent (in the Gregorian calendar) the first Monday in May, 2008.

Listing 4 Creating a date from components

```
NSDateComponents *components = [[NSDateComponents alloc] init];
```

```
[components setWeekday:2]; // Monday
[components setWeekdayOrdinal:1]; // The first day in the month
[components setMonth:5]; // May
[components setYear:2008];
NSCalendar *gregorian = [[NSCalendar alloc]
    initWithCalendarIdentifier:NSGregorianCalendar];
NSDate *date = [gregorian dateFromComponents:components];
```

Note that although you can configure an instance of `NSDateComponents` with “out of bounds” values for components (for example, in Listing 4 you could set the day component to `-6`), you should not rely on the result of this when *when creating a date from components*—the behavior may change from one release of the operating system to another.

Calendrical Calculations

`NSDate` provides the absolute scale and epoch for dates and times, which can then be rendered into a particular calendar, for calendrical calculations or user display. To perform calendrical calculations, you typically need to get the component elements of a date, such as the year, the month, and the day.

Adding Components to a Date

You use the `dateByAddingComponents:toDate:options:` method to add components of a date (such as hours or months) to an existing date. You can provide as many components as you wish. Listing 1 shows how to calculate a date an hour and a half in the future.

Listing 1 An hour and a half from now

```
NSDate *today = [[NSDate alloc] init];
NSDateComponents *offsetComponents = [[NSDateComponents alloc] init];
[offsetComponents setHour:1];
[offsetComponents setMinutes:30];
// Calculate when, according to Tom Lehrer, World War III will end
NSDate *endOfWorldWar3 = [gregorian dateByAddingComponents:comps toDate:today
options:0];
```

Components to add can be negative. The following example shows how you can get the Sunday in the current week (using a Gregorian calendar).

Listing 2 Getting the Sunday in the current week

```
NSDate *today = [[NSDate alloc] init];
NSCalendar *gregorian = [[NSCalendar alloc]
initWithCalendarIdentifier:NSGregorianCalendar];

// Get the weekday component of the current date
NSDateComponents *weekdayComponents = [gregorian components:NSWeekdayCalendarUnit
fromDate:today];

/*
Create a date components to represent the number of days to subtract from the
current date.
The weekday value for Sunday in the Gregorian calendar is 1, so subtract 1 from
the number of days to subtract from the date in question. (If today's Sunday,
subtract 0 days.)
*/
NSDateComponents *componentsToSubtract = [[NSDateComponents alloc] init];
[componentsToSubtract setDay: 0 - ([weekdayComponents weekday] - 1)];

NSDate *beginningOfWeek = [gregorian dateByAddingComponents:componentsToSubtract
toDate:today options:0];
```

```

/*
Optional step:
beginningOfWeek now has the same hour, minute, and second as the original date
(today).
To normalize to midnight, extract the year, month, and day components and create
a new date from those components.
*/
NSDateComponents *components =
    [gregorian components:(NSYearCalendarUnit | NSMonthCalendarUnit |
        NSDayCalendarUnit)
        fromDate: beginningOfWeek];
beginningOfWeek = [gregorian dateFromComponents:components];

```

The following example illustrates how you can calculate the first moment of the week (as defined by the calendar's locale):

Listing 3 Getting the beginning of the week

```

NSDate *today = [[NSDate alloc] init];
NSDate *beginningOfWeek = nil;
BOOL ok = [gregorian rangeOfUnit:NSWeekCalendarUnit startDate:&beginningOfWeek
    interval:NULL forDate:today];

```

Temporal Differences

You use `components:fromDate:toDate:options:` to determine the temporal difference between two dates in units other than seconds (which you could calculate with the `NSDate` method `timeIntervalSinceDate:`). Listing 4 shows how to get the number of months and days between two dates using a Gregorian calendar.

Listing 4 Getting the difference between two dates

```

NSDate *startDate = ...;
NSDate *endDate = ...;

NSCalendar *gregorian = [[NSCalendar alloc]
    initWithCalendarIdentifier:NSGregorianCalendar];

NSUInteger unitFlags = NSMonthCalendarUnit | NSDayCalendarUnit;

NSDateComponents *components = [gregorian components:unitFlags
    fromDate:startDate
    toDate:endDate options:0];

NSInteger months = [components month];
NSInteger days = [components day];

```

Converting from One Calendar to Another

To convert components of a date from one calendar to another—for example, from Gregorian to Hebrew—you first create a date object from the components using the first calendar, then you decompose the date into components using the second calendar. Listing 5 shows how to convert date components from one calendar to another.

Listing 5 Converting date components from one calendar to another

```
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setDay:6];
[comps setMonth:5];
[comps setYear:2004];

NSCalendar *gregorian = [[NSCalendar alloc]
                        initWithCalendarIdentifier:NSGregorianCalendar];
NSDate *date = [gregorian dateFromComponents:comps];
[comps release];
[gregorian release];

NSCalendar *hebrew = [[NSCalendar alloc]
                    initWithCalendarIdentifier:NSHebrewCalendar];
NSUInteger unitFlags = NSDayCalendarUnit | NSMonthCalendarUnit |
                    NSYearCalendarUnit;
NSDateComponents *components = [hebrew components:unitFlags fromDate:date];

NSInteger day = [components day]; // 15
NSInteger month = [components month]; // 9
NSInteger year = [components year]; // 5764
```


Using Time Zones

`NSTimeZone` is an abstract class that defines the behavior of time zone objects. Time zone objects represent geopolitical regions. Consequently, these objects have region names. Time zone objects also represent a temporal offset, either plus or minus, from Greenwich Mean Time (GMT) and an abbreviation (such as PST).

Creating Time Zones

Time zones affect the values of date components that are calculated by calendar objects for a given `NSDate` object. You can create an `NSTimeZone` object and use it to set the time zone of an `NSCalendar` object. By default, `NSCalendar` uses the application's (or process's) default time zone when the calendar object is created, which itself defaults to the time zone set in System Preferences.

`NSTimeZone` provides several class methods to make time zone objects: `timeZoneWithName:`, `timeZoneWithAbbreviation:`, and `timeZoneForSecondsFromGMT:`. The most flexible method is `timeZoneWithName:`. The name passed to this method may be in any of the formats understood by the system, for example EST, Etc/GMT-2, America/Argentina/Buenos_Aires, Europe/Monaco, or US/Pacific, as shown in the following code fragment:

```
NSTimeZone *timeZoneEST = [NSTimeZone timeZoneWithName:@"EST"];
NSTimeZone *timeZoneBuenos_Aires =
    [NSTimeZone timeZoneWithName:@"America/Argentina/Buenos_Aires"];
```

If you use `timeZoneWithAbbreviation:`, you can use only abbreviations such as EST. In the following code fragment, `timeZoneEST` will be initialized correctly, whereas `timeZoneUSPacific` will be `nil`.

```
NSTimeZone *timeZoneEST = [NSTimeZone timeZoneWithAbbreviation:@"EST"];
NSTimeZone *timeZoneUSPacific =
    [NSTimeZone timeZoneWithAbbreviation:@"US/Pacific"];
// timeZoneUSPacific = nil
```

For a complete list of time zone names and abbreviations known to the system, you can use the `knownTimeZoneNames` class method:

```
NSArray *timeZoneNames = [NSTimeZone knownTimeZoneNames];
```

Application Default Time Zone

You can set the default time zone within your application using `setDefaultTimeZone:`. You can access this default time zone at any time with the `defaultTimeZone` class method, and with the `localTimeZone` class method, you can get a relative time zone object that decodes itself to become the default time zone on any computer on which it finds itself.

Legacy API: NSDateCalendarDate

This appendix provides information about the class that represents a Gregorian calendar, `NSDateCalendarDate`. This class does not exist in iPhone OS and is deprecated in Mac OS X. All of the capabilities of `NSDateCalendarDate` are duplicated by other, recommended classes, as described elsewhere in this document.

Important: You should use instances of `NSDate` in conjunction with `NSDateComponents` to get the component elements of a date, and you should use an `NSDateFormatter` object to convert a date to a string. For the sake of legacy code that uses `NSDateCalendarDate`, the following sections describe using objects of that class to perform those same actions.

Using NSDateCalendarDate

A Gregorian date object, a special type of date object, is useful for representing dates users see. It adds methods for converting dates to strings, converting strings to dates, and retrieving elements from dates (such as hours, minutes, and the day of the week). This is implemented by `NSDateCalendarDate` in Objective-C and by `NSGregorianCalendar` in Java.

To retrieve conventional elements of a Gregorian date, use the `...of...` methods. For example, `dayOfWeek` returns a number that indicates the day of the week (0 is Sunday). The `monthOfYear` method returns a number between 1 and 12 that indicates the month. If you are using Mac OS X v10.4 or later, you can use a calendar object for more complicated calculations to determine, for example, how many weeks or months there are between two dates—see [“Calendars, Date Components, and Calendar Units”](#) (page 11).

You can use a Gregorian date instead (`NSDateCalendarDate` in Objective-C, `NSGregorianCalendar` in Java) to compare dates with less fine granularity than subsecond differences.

Note also that there are differences in the way that the `NSDateCalendarDate` Gregorian calendar (`NSGregorianCalendar`) and `NSDateCalendarDate` interpret components—for example, the `NSDateCalendarDate` Gregorian calendar’s week (interpreted using the `NSDateComponents` method `weekday`) starts with Sunday = 1, whereas the `NSDateCalendarDate` week (see `dayOfWeek`) starts with Sunday = 0.

Some Gregorian date methods return date objects that are automatically bound to time zone objects. These date objects use `NSTimeZone` to adjust dates for the proper locale. Unless you specify otherwise, objects returned from Gregorian date are bound to the default time zone for the current locale.

You can use the time zone associated with a date to change how the date prints its time interval. The time zone does not change how the time interval is stored. Because the value is stored independent of the time zone, you can accurately compare Gregorian dates with any other date objects or use them to create other date objects. It also means that you can track a date across different time zones; that is, you can create a new Gregorian date with a different time zone to see how the particular date is represented in that time zone.

String Representations of NSDate Objects

To represent your date object as an `NSString` object, use the `description...` methods. The simplest method, `description`, prints out the date in the format `YYYY-MM-DD HH:MM:SS ±HHMM`, where `±HHMM` represents the time zone offset in hours and minutes from GMT. (Adding the offset to the specific time yields the equivalent GMT.) To have a specific locale dictionary affect the representation of your `NSDate` object, use `descriptionWithLocale:` instead of `description`. The following keys in the locale dictionary affect `NSDate` objects:

Key	Description
<code>NSTimeDateFormatString</code>	Specifies how dates with times are printed. The default is to use full month names and days with a 24 hour clock, as in "Sunday, January 01, 2001 23:00:00 Pacific Standard Time."
<code>NSAMPMDesignation</code>	Specifies how the morning and afternoon designations are printed. The default is AM and PM.
<code>NSMonthNameArray</code>	Specifies the names for the months.
<code>NSShortMonthNameArray</code>	Specifies the abbreviations for the months.
<code>NSWeekDayNameArray</code>	Specifies the names for the days of the week.
<code>NSShortWeekDayNameArray</code>	Specifies the abbreviations for the days of the week.

Note that the `NSDate` implementation of the `description` method uses `NSDateCalendarDate`. `NSDateCalendarDate` does not model the transition from the Julian to the Gregorian calendar, so using the `description` method of `NSDate` yields inaccurate results for dates earlier than October 1582. If you need to describe dates earlier than the transition, you should use `NSDateFormatter` (as described in *Data Formatting Programming Guide for Cocoa*).

The Calendar Format

Each `NSDateCalendarDate` object has a calendar format associated with it. This format is a string that contains date-conversion specifiers that are very similar to those used in the standard C library function `strftime()`. `NSDateCalendarDate` interprets dates that are represented as strings conforming to this format. You can set the default format for an `NSDateCalendarDate` object at initialization time or using the `setCalendarFormat:` method. Several methods allow you to specify formats other than the one bound to the object.

The date conversion specifiers cover a range of date conventions:

Conversion specifier	Description
<code>%%</code>	A '%' character
<code>%a</code>	abbreviated name of the day of the week
<code>%A</code>	full name of the day of the week

Conversion specifier	Description
%b	abbreviated month name
%B	full month name
%c	shorthand for %X %x, the locale format for date and time
%d	day of the month as a decimal number (01-31)
%e	same as %d but does not print the leading 0 for days 1 through 9
%F	milliseconds as a decimal number (000-999)
%H	hour based on a 24-hour clock as a decimal number (00-23)
%I	hour based on a 12-hour clock as a decimal number (01-12)
%j	day of the year as a decimal number (001-366)
%m	month as a decimal number (01-12)
%M	minute as a decimal number (00-59)
%p	AM/PM designation for the locale
%S	second as a decimal number (00-59)
%w	day of the week as a decimal number (0-6), where Sunday is 0
%x	date using the date representation for the locale
%X	time using the time representation for the locale
%y	year without century (00-99)
%Y	year with century (such as 1990)
%Z	time zone name (such as Pacific Daylight Time)
%z	time zone offset in hours and minutes from GMT (HHMM)

Note that most of the formats that specify numeric values pad the width of the value with 0's (for example, %S represents 6 seconds as 06). You can suppress these 0's using a width specifier, just as you would with, for example, printf, as illustrated by the following example.

```

NSCalendarDate *date = [NSCalendarDate dateWithYear:2006 month:2 day:2
                                hour:2 minute:2 second:2
                                timeZone:nil];
[date setCalendarFormat:@"%m (%1m), %d (%1d), %I (%1I), %j (%1j), %S (%1S)"];
NSLog(@"date: %@", date);

// Output: date: 02 (2), 02 (2), 02 (2), 033 (33), 02 (2)

```

String Representations of NSDate Objects

`NSDate` provides several `description...` methods for representing dates as strings. These methods—`description`, `descriptionWithLocale:`, `descriptionWithCalendarFormat:`, and `descriptionWithCalendarFormat:locale:`—take an implicit or explicit calendar format. The user's locale information affects the returned string. `NSDate` accesses the locale information as an `NSDictionary` object. If you use `descriptionWithLocale:` or `descriptionWithCalendarFormat:locale:`, you can specify a different locale dictionary. The following keys in the locale dictionary affect `NSDate`:

Locale key	Description
<code>NSTimeDateFormatString</code>	Specifies how dates with times are printed, affecting strings that use the format specifiers <code>%C</code> , <code>%X</code> , or <code>%x</code> . The default is to use abbreviated months and days with a 24 hour clock, as in "Sun Jan 01 23:00:00 +6 2001".
<code>NSAMPMDesignation</code>	Specifies how the morning and afternoon designations are printed, affecting strings that use the <code>%p</code> format specifier. The default is AM and PM.
<code>NSMonthNameArray</code>	Specifies the names for the months, affecting strings that use the <code>%B</code> format specifier.
<code>NSShortMonthNameArray</code>	Specifies the abbreviations for the months, affecting strings that use the <code>%b</code> format specifier.
<code>NSWeekDayNameArray</code>	Specifies the names for the days of the week, affecting strings that use the <code>%A</code> format specifier.
<code>NSShortWeekDayNameArray</code>	Specifies the abbreviations for the days of the week, affecting strings that use the <code>%a</code> format specifier.

If you subclass `NSDate` and override `description`, you should also override `descriptionWithLocale:`. The `stringWithFormat:` method of `NSString` uses `descriptionWithLocale:` instead of `description` when you use the `%@` conversion specifier to print an `NSDate`. That is, this message:

```
[NSString stringWithFormat:@"The current date and time are %@",
    [MyNSDateSubclass date]]
```

invokes `descriptionWithLocale:`.

Document Revision History

This table describes the changes to *Date and Time Programming Guide for Cocoa*.

Date	Notes
2009-07-21	Added links to Cocoa Core Competencies.
2008-07-03	Moved information about <code>NSDate</code> to an appendix, rewrote articles to replace references to <code>NSDate</code> , and expanded content.
	Added a section about how to get date components using <code>NSDate</code> and <code>NSDateComponents</code> and a section about how to convert from one calendar to another.
	Removed information about converting a date to a string. See <i>NSDateFormatter Class Reference</i> for that information.
2007-09-04	Enhanced discussion of calendrical calculations using <code>NSDateComponents</code> .
2007-03-06	Added note regarding Julian and Gregorian calendars.
2006-05-23	Corrected typographical errors. Added a note about the use of width specifiers for calendar date format strings.
2006-02-07	Updated to include <code>NSDate</code> and <code>NSDateFormatter</code> changes introduced in Mac OS X v10.4.
2005-08-11	Changed title from "Dates and Times." Corrected minor typographic error.
2002-11-12	Revision history was added to existing document. It will be used to record changes to the content of the document.

