NSDate Class Cluster

Class Cluster Description

NSDate objects represent a single point in time. The NSDate cluster's single public superclass, NSDate, declares the programmatic interface for specific and relative time values.

The objects you create using NSDate are referred to as *date objects*. They are immutable objects. Because of the nature of class clusters, objects returned by the NSDate class are not instances of that abstract class but of one of its private subclasses. Although a date object's class is private, its interface is public, as declared by the abstract superclass, NSDate. (See "Class Clusters" in the introduction to the Foundation Kit for more information on class clusters and creating subclasses within a cluster.)

Generally, you instantiate a suitable date object by invoking one of the **date...** class methods.

The date classes adopt the NSCopying and NSCoding protocols.

NSDate

Inherits From: NSObject

Conforms To: NSCoding, NSCopying
NSObject (NSObject)

Declared In: Foundation/NSDate.h

Class at a Glance

Purpose

An NSDate object stores a date and time that can be compared to other dates and times.

Principal Attributes

- Seconds since absolute reference date (1 January, 2001, GMT)

Creation

+ date	Represents the current date and time.
+ dateWithTimeIntervalSinceNow:	Represents a date relative to the current date and time.
+ dateWithTimeIntervalSinceReferenceDate	Represents a date relative to the absolute reference date.

Commonly Used Methods

– earlierDate:	Compares the receiver to the argument and returns the earlier of the two.
– isEqual:	Returns YES if the receiver and the argument are equal.
– laterDate:	Compares the receiver to the argument and returns the later of the two.
– timeIntervalSinceNow	Returns the number of seconds difference between the receiver and the current date and time.

Primitive Method

– timeIntervalSinceReferenceDate

Class Description

NSDate is an abstract class that provides behavior for creating dates, comparing dates, representing dates, computing intervals, and similar functionality. It presents a programmatic interface through which suitable date objects are requested and returned. Date objects returned from NSDate are lightweight and immutable since they represent a invariant point in time. This class is designed to provide the foundation for arbitrary calendrical representations. Its subclass NSCalendarDate offers date objects that are suitable for representing dates according to western calendrical systems.

"Date" as used above implies clock time as well. The standard unit of time for date objects is a value typed as NSTimeInterval and expressed as seconds. The NSTimeInterval type makes possible a wide and fine-grained range of date and time values, giving accuracy within milliseconds for dates 10,000 years apart.

NSDate and its subclasses compute time as seconds *relative* to an absolute reference date. This reference date is the first instant of 1 January, 2001, Greenwich Mean Time (GMT). NSDate converts all date and time representations to and from NSTimeInterval values that are relative to this absolute reference date. A positive interval relative to a date represents a point in the future, a negative interval represents a time in the past.

Note: Conventional UNIX systems implement time according to the Network Time Protocol (NTP) standard, which is based on Coordinated Universal Time. The current private implementations of NSDate follow the NTP standard. However, they do not account for leap seconds and therefore are not synchronized with International Atomic Time (the most accurate).

Like various other Foundation classes, NSDate enables you to obtain operating-system functionality (dates and times) without depending on operating-system internals. It also provides a basis for the NSRunLoop and NSTimer classes, which use concrete date objects to implement local event loops and timers.

NSDate's sole primitive method, **timeIntervalSinceReferenceDate**, provides the basis for all the other methods in the NSDate interface. It returns a time value relative to an absolute reference date.

Creating NSDate Objects

Use a date object to store a point in time. If you want to store the current time, use the **date** class method to create the date object. If you want to store some time other than the current time, use one of the **dateWithTimeInterval...** methods.

The **dateWithTimeInterval...** methods create 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. The code fragment below defines two date objects. **tomorrow** is exactly 24 hours from the current date and time, and **yesterday** is exactly 24 hours earlier than the current date and time.

```
NSTimeInterval secondsPerDay = 24 * 60 * 60;
NSDate *tomorrow = [NSDate
          dateWithTimeIntervalSinceNow:secondsPerDay];
NSDate *yesterday = [NSDate
          dateWithTimeInstervalSinceNow:-secondsPerDay];
```

To get new date objects with date-and-time values adjusted from existing date objects, use **addTimeInterval:**.

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

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

NSCalendarDate

The NSDate class cluster provides, for your convenience, a public concrete subclass of NSDate that satisfies many requirements for dates and times. This subclass, NSCalendarDate, enables you to represent dates as arbitrary strings, to create new date objects from string representations, to extract date and time elements from date objects, and to do other calendar-related functions. You can create an NSCalendarDate out of your NSDate using the **dateWithCalendarFormat:timeZone:** method.

Comparing NSDate Objects

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

To compare dates, use the **isEqual:**, **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 or use NSCalendarDate objects instead. The following code 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) ...
```

NSString Representations for NSDates

To represent your date object as an NSString, 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,

use **descriptionWithLocale:** instead of **description**. The following keys in the locale dictionary affect NSDates:

NSTimeDateFormatString	Specifies how dates with times are printed. The default is to use abbreviated months and days with a 24 hour clock, as in "Sun Jan 01 23:00:00 +6 2001."
NSAMPMDesignation	Specifies how the morning and afternoon designations are printed. The default is AM and PM.
NSMonthNameArray	Specifies the names for the months.
NSShortMonthNameArray	Specifies the abbreviations for the months.
NSWeekDayNameArray	Specifies the names for the days of the week.
NSShortWeekDayNameArray	Specifies the abbreviations for the days of the week.

Subclassing NSDate

If you want to subclass NSDate to obtain behavior different than that provided by the private subclasses, you must do these three things:

- Declare a suitable instance variable to hold the date and time value (relative to an absolute reference date).
- Override the **timeIntervalSinceReferenceDate** instance method to provide the correct date and time value based on your instance variable.
- Override **initWithTimeIntervalSinceReferenceDate:**, the designated initializer method.

Your subclass may use a different reference date than the absolute reference date used by NSDate (the first instance of 1 January 2001 GMT). If it does, it must still use the absolute reference date in its implementations of the methods **timeIntervalSinceReferenceDate** and **initWithTimeSinceReferenceDate:**. That is, the reference date referred to in the titles of these methods is the absolute reference date. If you do not use the absolute reference date in these methods, comparisons between NSDate objects of your subclass and NSDate objects of a private subclass will not work.

Adopted Protocols

NSCopying	– copyWithZone:
NSCoding	– encodeWithCoder: – initWithCoder:

Method Types

Creating an NSDate instance	+ date + dateWithTimeIntervalSinceNow: + dateWithNaturalLanguageString: + dateWithNaturalLanguageString:locale: + dateWithTimeIntervalSinceReferenceDate: + dateWithTimeIntervalSince1970: + distantFuture + distantPast + allocWithZone: – initWithTimeIntervalSinceNow: – initWithString: – initWithTimeInterval:sinceDate: – init – initWithTimeIntervalSinceReferenceDate: – addTimeInterval:
Comparing dates	– isEqualToDate: – earlierDate: – laterDate: – compare:
Getting time intervals	– timeIntervalSinceDate: – timeIntervalSinceNow + timeIntervalSinceReferenceDate – timeIntervalSinceReferenceDate – timeIntervalSince1970
Representing dates as NSStrings	– description – descriptionWithLocale: – descriptionWithCalendarFormat:timeZone:locale:
Converting to an NSCalendarDate object	– dateWithCalendarFormat:timeZone:

Class Methods

allocWithZone:

+ (id)**allocWithZone**:(NSZone *)*zone*

Allocates an uninitialized NSDate in zone. **nil** is returned if allocation fails. Usually, you use **date**, **dateWithTimeIntervalSinceNow:**, **dateWithTimeIntervalSince1970:**, or

dateWithTimeIntervalSinceReferenceDate: instead of this method. You are responsible for releasing objects created with **alloc**... methods.

See also: **+ date**, **+ dateWithTimeIntervalSinceNow**:, **+ dateWithTimeIntervalSince1970:**, **+ dateWithTimeIntervalSinceReferenceDate:**

date

+ (NSDate *)**date**

Creates and returns an NSDate set to the current date and time. This method uses the default initializer method for the class, **init**.

A typical example of using **date** to get the current date is:

```
NSDate *today = [NSDate date];
```

See also: **– init**

NS+ dateWithNaturalLanguageString:

+ (NSDate *)**dateWithNaturalLanguageString:**(NSString *)*string*

Creates and returns an NSDate set to the date and time specified by *string*. The argument *string* can be a colloquial specification of a date, such as “last Tuesday at dinner,” “3pm December 31, 1995,” “12/31/95,” or “31/12/95.” In parsing *string*, this method uses the date and time preferences stored in the user’s defaults database. (See **dateWithNaturalLanguageString:locale:** for a list of the specific items used.)

See also: + **dateWithNaturalLanguageString:locale:**

NS+ dateWithNaturalLanguageString:locale:

+ (NSDate *)**dateWithNaturalLanguageString:**(NSString *)*string*
locale:(NSDictionary *)*localeDictionary*

Creates and returns an NSDate set to the date and time specified by *string*. The argument *string* can be a colloquial specification of a date, such as “last Tuesday at dinner,” “3pm December 31, 1995,” “12/31/95,” or “31/12/95.” The keys and values that represent the locale data from *localeDictionary* are used when

parsing the string. In addition to the locale keys listed in the class description, these keys are used when parsing natural language strings:

NSDateTimeOrdering	Determines how to use ambiguous numbers. Specify this value as a permutation of the letters M (month), D (day), Y (year), and H (hour). For example, MDYH treats "2/3/95 10" as the 3rd day of February 1995 at 10:00am, whereas DMYH treats the same value as the 2nd day of March 1995 at 10:00am. If fewer numbers are specified than are needed, the numbers are prioritized to satisfy day first, then the month, and then the year. For example, if you supply only the value 12, it means the 12th day of this month in this year because the day must be specified. If you supply "2 12" it means either February 12 or December 2, depending on if the ordering is "MDYH" or "DMYH."
NSEarlierTimeDesignations	An array of strings that denote a time in the past. These are adjectives that modify values from NSYearMonthWeekDesignations. The defaults are "prior," "last," "past," and "ago."
NSHourNameDesignations	Strings that identify the time of day. These strings should be bound to an hour. The default is this array of arrays: (0, midnight), (12, noon, lunch), (10, morning), (14, afternoon), (19, dinner).
NSLaterTimeDesignations	An array of strings that denote a time in the future. This is an adjective that modifies a value from NSYearMonthWeekDesignations. The default is "next."
NSNextDayDesignations	A string that identifies the day after today. The default is "tomorrow."
NSNextNextDayDesignations	A string that identifies the day after tomorrow. The default is "nextday".
NSPriorDayDesignations	A string that identifies the day before today. The default is "yesterday."
NSThisDayDesignations	A string that identifies what this day is called. The default is "today."
NSYearMonthWeekDesignations	An array of strings that specify the word for year, month, and week in the current locale. The defaults are "year," "month," and "week."

See also: + **dateWithNaturalLanguageString:**

NS+ dateWithString:

+ (id)**dateWithString:**(NSString *)*aString*

Creates and returns an NSDate with a date and time value specified by the international string-representation format: YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM is a time zone offset

in hours and minutes from GMT (for example, “1994-05-23 10:45:32 +0600”). You must specify all fields of the format, including the time zone offset, which must have a plus- or minus-sign prefix.

See also: – **initWithString:**

dateWithTimeIntervalSinceNow:

+ (NSDate *)**dateWithTimeIntervalSinceNow**:(NSTimeInterval)*seconds*

Creates and returns an NSDate set to the specified number of seconds from the current date and time. Use a negative argument value to specify a date before the current date.

See also: – **initWithTimeIntervalSinceNow:**

dateWithTimeIntervalSince1970:

+ (NSDate *)**dateWithTimeIntervalSince1970:**(NSTimeInterval)*seconds*

Creates an returns an NSDate set to the specified number of seconds from the reference date used by UNIX systems, 1 January 1970. Use a negative argument to specify a date before this date.

See also: – **timeIntervalSince1970**

dateWithTimeIntervalSinceReferenceDate:

+ (NSDate *)**dateWithTimeIntervalSinceReferenceDate**:(NSTimeInterval)*seconds*

Creates and returns an NSDate set to a specified number of seconds from the absolute reference date (the first instant of 1 January 2001, GMT). Use a negative argument to specify a date and time before the reference date.

See also: – **initWithTimeIntervalSinceReferenceDate:**

distantFuture

+ (NSDate *)**distantFuture**

Creates and returns an NSDate that represents a date in the distant future (in terms of centuries).

You can pass this value where an NSDate is required to have the date argument essentially ignored. For example, the NSWindow method **nextEventMatchingMask:untilDate:inMode:dequeue:** returns **nil** if an event specified in the event mask does not happen before the specified date. You can use the object returned by **distantFuture** as the date argument to wait indefinitely for the event to occur.

```
myEvent = [myWindow nextEventMatchingMask:myEventMask
    untilDate:[NSDate distantFuture]
```

```
    inMode:NSDefaultRunLoopMode
    dequeue:YES];
```

See also: + **distantPast**

distantPast

+ (NSDate *)**distantPast**

Creates and returns an NSDate that represents a date in the distant past (in terms of centuries). You can use this object in your code as a control date, a guaranteed temporal boundary.

See also: + **distantFuture**

timeIntervalSinceReferenceDate

+ (NSTimeInterval)**timeIntervalSinceReferenceDate**

Returns the interval between the system's absolute reference date (the first instance of 1 January 2001, GMT) and the current date and time. Currently, this value is a negative number.

See also: – **timeIntervalSinceReferenceDate**, – **timeIntervalSinceDate:**, – **timeIntervalSince1970**, – **timeIntervalSinceNow**

Instance Methods

addTimeInterval:

– (id)**addTimeInterval:**(NSTimeInterval)*seconds*

Returns an NSDate object that is set to a specified number of seconds relative to the receiver. Use a negative value for *seconds* to have the returned object specify a date before the receiving object. The date returned might have a representation different from the receiver's.

See also: – **initWithTimeInterval:sinceDate:**, – **timeIntervalSinceDate:**

compare:

– (NSComparisonResult)**compare:**(NSDate *)*anotherDate*

Compares the receiving date to *anotherDate*, using **timeIntervalSinceDate:**, and returns a value of type NSComparisonResult. If the two dates are exactly equal to each other, this method returns NSOrderedSame. If the receiving object in the comparison is more recent than *anotherDate*, the method returns NSOrderedDescending. If it is older, it returns NSOrderedAscending.

This method detects subsecond differences between dates. If you want to compare dates with a less fine granularity, use **timeIntervalSinceDate:** to compare the two dates or use NSCalendarDate objects instead.

See also: – **earlierDate:**, – **isEqual**: (NSObject protocol), – **laterDate:**

dateWithCalendarFormat:timeZone

– (NSCalendarDate *)**dateWithCalendarFormat**:(NSString *)*formatString*
timeZone:(NSTimeZone *)*timeZone*

Converts the NSDate to an NSCalendarDate object bound to *formatString* and the time zone *timeZone*. If you specify **nil** for either or both of these arguments, the default format string and time zone are assumed. (The default time zone is the one specific to the current locale; the default format string, which is "%Y-%m-%H:%M:%S %z", conforms to the international format YYYY-MM-DD HH:MM:SS ±HHMM.)

The conversion specifiers for *formatString* cover a range of date conventions:

%%	a '%' character
%a	abbreviated weekday name
%A	full weekday name
%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)
%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 (00-61)
%w	weekday 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 abbreviation (such as PDT)
%z	time zone offset in hours and minutes from GMT (HHMM)

See also: – **description**, – **descriptionWithCalendarFormat:timeZone:locale:**, – **descriptionWithLocale:**, + **dateWithString:calendarFormat:** (NSCalendarDate)

description

– (NSString *)**description**

Returns an NSString representing the NSDate object and conforming to the international format YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM represents the time zone offset in hours and minutes from GMT (for example, “1994-05-23 10:45:32 +0600”).

See also: – **dateWithCalendarFormat:timeZone:**,
– **descriptionWithCalendarFormat:timeZone:locale:**, – **descriptionWithLocale:**,
– **description** (NSCalendarDate)

descriptionWithCalendarFormat:timeZone:locale:

– (NSString *)**descriptionWithCalendarFormat:**(NSString *)*formatString*
timeZone:(NSTimeZone *)*aTimeZone*
locale:(NSDictionary *)*localeDictionary*

Returns an NSString representing the NSDate, formatted as specified by the conversion specifiers in *formatString* (see the method description for **dateWithCalendarFormat:timeZone:** for a list of these). Specify the time zone for the date in *aTimeZone* and specify keys and values representing the locale in *localeDictionary*. (See the class description for a list of the appropriate locale dictionary keys.) If you specify **nil** for one of these arguments, its default value is assumed. (The default time zone is the one specific to the current locale; the default format string, which is “%Y-%m-%H:%M:%S %z”, conforms to the international format YYYY-MM-DD HH:MM:SS ±HHMM.)

You could use this method to print the current time as follows:

```
sprintf(aString, "The current time is %s\n", [[[NSDate date]
   descriptionWithCalendarFormat:@"%H:%M:%S %Z" timeZone:nil
   locale:nil] cString]);
```

See also: – **description**, – **descriptionWithCalendarFormat:locale:** (NSCalendarDate),
– **descriptionWithLocale:**

descriptionWithLocale:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*localeDictionary*

Returns an NSString representing the NSDate that conforms to the international format YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM represents the time zone offset in hours and minutes from GMT (for example, “1994-05-23 10:45:32 +0600”). Included are the keys and values that represent the locale data from *localeDictionary* (see the class description).

See also: – **description**, – **descriptionWithCalendarFormat:timeZone:locale:**,
– **descriptionWithLocale:** (NSCalendarDate)

earlierDate:

– (NSDate *)**earlierDate:**(NSDate *)*anotherDate*

Compares the receiver date to *anotherDate*, using **timeIntervalSinceDate:**, and returns the earlier of the two.

See also: – **compare:**, – **isEqual**: (NSObject protocol), – **laterDate:**

init

– (id)**init**

When sent to the object returned by **alloc** or **allocWithZone:**, this method returns an NSDate initialized to the current date and time. This method uses the designated initializer, **initWithTimeIntervalSinceReferenceDate:**.

See also: + **date**, – **initWithTimeIntervalSinceReferenceDate:**

initWithString:

– (id)**initWithString:**(NSString *)*description*

Returns an NSDate with a date and time value specified by the international string-representation format: YYYY-MM-DD HH:MM:SS ±HHMM, where ±HHMM is a time zone offset in hours and minutes from GMT (for example, “1994-05-23 10:45:32 +0600”). You must specify all fields of the format, including the time zone offset, which must have a plus- or minus-sign prefix. This method uses the designated initializer, **initWithTimeIntervalSinceReferenceDate:**.

See also: – **dateWithString:**, – **description**, – **initWithTimeIntervalSinceReferenceDate:**

initWithTimeInterval:sinceDate:

– (NSDate *)**initWithTimeInterval:**(NSTimeInterval)*seconds* **sinceDate:**(NSDate *)*anotherDate*

Returns an NSDate initialized relative to another NSDate by a specified number of *seconds* (plus or minus). This method uses the designated initializer, **initWithTimeIntervalSinceReferenceDate:**.

See also: – **initWithTimeIntervalSinceReferenceDate:**

initWithTimeIntervalSinceNow:

– (NSDate *)**initWithTimeIntervalSinceNow:**(NSTimeInterval)*seconds*

Returns an NSDate initialized relative to the current date and time by the specified number of *seconds* (plus or minus). This method uses the designated initializer, **initWithTimeIntervalSinceReferenceDate:**.

See also: + **dateWithTimeIntervalSinceNow:**, – **initWithTimeIntervalSinceReferenceDate:**

initWithTimeIntervalSinceReferenceDate:

– (id)**initWithTimeIntervalSinceReferenceDate:**(NSTimeInterval)*seconds*

When sent to the object returned by **alloc** or **allocWithZone:**, this method returns an NSDate initialized relative to the absolute reference date (the first instant of 1 January 2001, GMT) by the specified number of seconds (plus or minus).

This method is the designated initializer for the NSDate class and is declared primarily for the use of subclasses of NSDate. When you subclass NSDate to create a concrete date class, you must override this method.

See also: + **dateWithTimeIntervalSinceReferenceDate:**

isEqualToDate:

– (BOOL)**isEqualToDate:**(id)*anotherDate*

Returns YES if the two objects compared are NSDate objects and are exactly equal to each other, NO if one of the objects is not of the NSDate class or if their date and time values differ. This method detects subsecond differences between dates. If you want to compare dates with a less fine granularity, either use **timeIntervalSinceDate:** to compare the two dates or use NSCalendarDate objects instead.

See also: **– compare:**, – **earlierDate**:, – **laterDate:**

laterDate:

– (NSDate *)**laterDate:**(NSDate *)*anotherDate*

Compares the receiver to *anotherDate*, using **timeIntervalSinceDate:**, and returns the later of the two.

See also: – **compare:**, – **earlierDate:**, – **isEqual**: (NSObject protocol)

timeIntervalSinceDate:

– (NSTimeInterval)**timeIntervalSinceDate:**(NSDate *)*anotherDate*

Returns the interval between the receiver and *anotherDate*. If the return value is a negative number, *anotherDate* is earlier than the receiver.

See also: – **timeIntervalSince1970**, – **timeIntervalSinceNow**, – **timeIntervalSinceReferenceDate**

timeIntervalSince1970

– (NSTimeInterval)**timeIntervalSince1970**

Returns the interval between the receiver and the reference date used by UNIX systems, 1 January 1970. If the receiver is earlier than 1 January 1970, the return value is negative.

See also: – **timeIntervalSinceDate:**, – **timeIntervalSinceNow**, – **timeIntervalSinceReferenceDate**

timeIntervalSinceNow

– (NSTimeInterval)**timeIntervalSinceNow**

Returns the interval between the receiver and the current date and time. If the receiver is earlier than the current date and time, the return value is negative.

See also: – **timeIntervalSinceDate:**, – **timeIntervalSince1970**, – **timeIntervalSinceReferenceDate**

timeIntervalSinceReferenceDate

– (NSTimeInterval)**timeIntervalSinceReferenceDate**

Returns the interval between the receiver and the system's absolute reference date, 1 January 2001, GMT. If the receiver is earlier than the absolute reference date, the return value is negative.

This is the primitive method for NSDate. If you subclass NSDate, you must override this method with your own implementation for it.

See also: – **timeIntervalSinceDate:**, – **timeIntervalSince1970**, – **timeIntervalSinceNow**, + **timeIntervalSinceReferenceDate**