NSDate Class Cluster

Class Cluster Description

Through the NSDate class cluster you can obtain immutable objects that represent a single point in time. NSDate, an abstract class, is the programmatic interface to private concrete classes. Through NSDate you can obtain date objects for specific and relative time values.

The objects you create using NSDate are referred to as *date objects*. Because of the nature of class clusters, date 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 ªClass Clustersº 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: NSObject
NSCopying

Declared In: foundation/NSDate.h

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.

ªDateº as used above implies clock time as well. The standard unit of time for date objects is a value typed as NSTimeInterval (currently a **double**) 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, 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 other Foundation classes, NSDate enables you to obtain operating-system functionality (dates and times) without depending on operating-system internals. It also lays a foundation for the NSRunLoop and NSTimer classes, which use concrete date objects to implement local event loops and timers. Run loop objects manage input from ports (NSPort) and timers; timers send action messages to targets at a specific interval or recurrent intervals.

The central methods of NSDate are **initWithTimeIntervalSinceReferenceDate**, **init**, and **timeIntervalSinceReferenceDate** and the NSCopying protocol method **copyWithZone:**. These methods provide the basis for all the other methods in the NSDate interface. The single primitive method, **timeIntervalSinceReferenceDate**, returns a constant time value relative to an absolute reference date.

Using NSDate

The date objects you obtain through NSDate give you a diverse range of date and time functionality. To obtain dates, send one of the **date...** class methods to NSDate. One of the most useful is **date** itself, which returns a date object representing the current date and time. You can get new date objects with date-and-time values adjusted from existing date objects by sending **addTimeInterval:**.

You can obtain relative date information by sending the **timeInterval...** methods to a date object. For instance, **timeIntervalSinceNow** gives you the time, in seconds, between the current time and the receiving date object. Compare dates with the **isEqual:**, **compare: laterDate:** and **earlierDate:** methods and use the **description** method to obtain a string object that represents the date in a standard international format.

NSCalendarDate

The NSDate class cluster provides, for your convenience, a public concrete subclass of NSDate that will satisfy 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.

Subclassing NSDate

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

- Declare a suitable instance variable to hold the date and time value (relative to an absolute reference date).
- Override the **timeIntervalSinceReferenceDate** class and primitive methods to return this value.
- Initialize the date-and-time instance variable to an appropriate value by overriding the initialization methods you want for your class. For **init**, this should be the current date and time. Note that **init** and **initWithTimeIntervalSinceReferenceDate:** are designated initializers.

Instance Variables

None declared in this class.

Adopted Protocols

NSCopying	- copyWithZone: - copy
NSCoding	- encodeUsingCoder: - decodeUsingCoder:

Method Types

Allocating and initializing an NSDate object

+ allocWithZone:
+ date

	+ dateWithTimeIntervalSinceNow: + dateWithTimeIntervalSinceReferenceDate: + distantFuture + distantPast -ð init - initWithString: - initWithTimeInterval:sinceDate: - initWithTimeIntervalSinceNow: - initWithTimeIntervalSinceReferenceDate:
Converting to an NSCalendar object	- dateWithCalendarFormat:timeZone:
Representing dates	- description - descriptionWithCalendarFormat:timeZone:
Adding and getting intervals	- addTimeInterval: - timeIntervalSinceDate: - timeIntervalSinceNow + timeIntervalSinceReferenceDate - timeIntervalSinceReferenceDate
Comparing dates	- compare: - earlierDate: - isEqual: - laterDate:

Class Methods

allocWithZone:

+ **allocWithZone**:(NXZone *)*zone*

Allocates an unitialized instance of a concrete date from the specified *zone*. If allocation fails, **nil** is returned.

Typically, you create dictionary objects using the **date...** class methods, not the **alloc...** and **init...** methods. Note that it's your responsibility to release (with either **release** or **autorelease**) those objects created with the **alloc...** methods.

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

date

+ (NSDate *)**date**

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

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

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

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

dateWithTimeIntervalSinceNow:

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

Creates and returns an instance of NSDate set to a specified number of seconds before or after the current date and time.

```
NSDate *overTime = [NSDate dateWithTimeIntervalSinceNow:8*360];
```

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

dateWithTimeIntervalSinceReferenceDate:

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

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

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

distantFuture

+ (NSDate *)**distantFuture**

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

See also: + **distantPast**

distantPast

+ (NSDate *)**distantPast**

Creates and returns a date object 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 and the current date and time. This value is less than zero until the first instant of 1 January 2001, GMT.

See also: - **timeIntervalSinceDate:**;- **timeIntervalSinceReferenceDate**; - **timeIntervalSinceNow**

Instance Methods

addTimeInterval:

- (NSDate *)**addTimeInterval:(**NSTimeInterval**)***seconds*

Returns an NSDate object that is set to a specified number of seconds relative to the receiver. The **NSTimeInterval** argument value can be positive or negative (positive meaning later). The date returned might have a representation different from the receiver's.

See also: - **timeIntervalSinceDate:**

compare:

- (NSComparisonResult)**compare:**(NSDate *)*otherDate*

Compares the receiving date to *otherDate* and returns a value of type NSComparisonResult. If the receiving object in the comparison is more recent than *otherDate*, the method returns NSOrderedDescending. If it is older, it returns NSOrderedAscending. If they are equal, it returns NSOrderedSame.

See also: - **earlierDate:**, - **isEqual**:, - **laterDate:**

dateWithCalendarFormat:timeZone

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

Converts the date object to an NSCalendarDate object bound to the format string *formatString* and the time zone *timeZone*. If you specify **nil** after 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 ª%Y-%m-%H:%M:%S %zº, conforms to the international format YYYY-MM-DD HH:MM:SS -HHMM.) The date-conversion specifiers cover a range of date conventions. See the description of the class method **dateWithString:calendarFormat:** in the NSCalendarDate class specification for a listing of these specifiers.

See also: - **description**

description

- (NSString *)**description**

Returns a string representation of the NSDate object 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 Greenwich Mean Time (GMT). An example might be ª1994-05-23 10:45:32 +0600º.

See also: - **descriptionWithCalendarFormat:timeZone:**

descriptionWithCalendarFormat:timeZone

- (NSString *)**descriptionWithCalendarFormat:**(NSString *)*format* **timeZone:**(NSTimezone *)*aTimeZone*

Returns a string representation of the date object that is formatted as specified by the conversion specifiers in the format string *format*. Specify the time zone for the date in *aTimeZone*. The conversion specifiers 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)

%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 associated with a 12-hour clock
%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 (e.g. 1990)
%Z	time zone name
%z	timezone offset in hours and minutes from GMT (HHMM)

See also: - **description,** - **descriptionWithCalendarFormat:timeZone:** (NSCalendarDate)

init

- **init**

When sent to the object returned by **alloc** or **allocWithZone:**, this method returns an initialized date object. This method is the designated initializer for the NSDate class and is declared primarily for the use of subclasses of NSDate. The designated initializer of the subclass should, through a message to **super**, invoke this method. Returns **self**.

When you subclass NSDate to create a concrete date class, you must override this method. When you override this method, allocate an instance of your class and set its date-and-time instance variable to an appropriate value, such as the current date and time.

initWithString:

- **initWithString:**(NSString *)*description*

Returns an calendar date object 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

Greenwich Mean Time. (Adding the offset to the specified time yields the equivalent GMT.) An example string might be ª1994-03-30 13:12:43 +0900º. You must specify all fields of the format, including the time-zone offset, which must have a plus- or minus-sign prefix.

See also: - **description**

initWithTimeInterval:sinceDate:

- (NSDate *)**initWithTimeInterval:**(NSTimeInterval)*secsToBeAdded* **sinceDate:**(NSDate *)*anotherDate*

Returns an NSDate object initialized relative to another date object by a specified number of seconds (plus or minus). If you have not overridden NSDate's initWithTimeIntervalSinceReferenceDate: in your subclass, this method generates an exception message and returns nil. Your conversions between two dates using the same representation should be exact.

```
NSDate *nextquarter = [[NSDate alloc]
                      initWithTimeInterval:(86400.0*365.25)/4.0
                      sinceDate:[NSDate date]]
```

initWithTimeIntervalSinceNow:

- (NSDate *)**initWithTimeIntervalSinceNow:**
  (NSTimeInterval)*secsToBeAddedToNow*

Returns an NSDate object initialized relative to the current date and time by a specified number of seconds (plus or minus). If you have not overridden NSDate's **initWithTimeIntervalSinceReferenceDate:** in your subclass, this method raises an exception.

initWithTimeIntervalSinceReferenceDate:

- **initWithTimeIntervalSinceReferenceDate:**(NSTimeInterval)*secsToBeAdded*

When sent to the object returned by **alloc** or **allocWithZone:**, this method returns an initialized date object. This method is the designated initializer for the NSDate class and is declared primarily for the use of subclasses of NSDate. The designated initializer of the subclass should, through a message to **super**, invoke this method.

When you subclass NSDate to create a concrete date class, you must override this message. In doing so, allocate an instance of your class and set its NSTimeInterval instance variable to an appropriate value, such as the current date and time adjusted by *secsToBeAdded*.

isEqual:

- (BOOL)**isEqual:***other*

Returns YES if the two objects compared are NSDate objects and are within one second of each other, NO if the objects are both not of the NSDate class or if they differ by more than one second. If you want to detect sub-second differences, send **timeIntervalSinceReferenceDate** to both objects and subtract the values returned.

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

earlierDate:

- (NSDate *)**earlierDate:**(NSDate *)*otherDate*

Compares the receiver date to *otherDate* and returns the older of the two.

See also: - **compare:**, - **isEqual**:, - **laterDate:**

laterDate:

- (NSDate *)**laterDate:**(NSDate *)*otherDate*

Compares the receiver date to *otherDate* and returns the later of the two.

See also: - **compare:**, - **earlierDate:**, - **isEqual**:

timeIntervalSinceDate:

- (NSTimeInterval)**timeIntervalSinceDate:**(NSDate *)*otherDate*

Returns the interval between the receiver and *otherDate*. This method uses **timeIntervalSinceReferenceDate**.

See also: - **timeIntervalSinceNow**, - **timeIntervalSinceReferenceDate**

timeIntervalSinceNow

- (NSTimeInterval)**timeIntervalSinceNow**

Returns the interval between the receiver and the current date and time (which is positive for future dates). This

method uses **timeIntervalSinceReferenceDate**.

See also: - **timeIntervalSinceDate:**, - **timeIntervalSinceReferenceDate**

timeIntervalSinceReferenceDate

- (NSTimeInterval)**timeIntervalSinceDate**

Returns the interval between the receiver and the system's absolute reference date. This value is less than zero until the first instant of 1 January 2001, GMT.

See also: - **timeIntervalSinceDate:**, - **timeIntervalSinceNow**, + **timeIntervalSinceReferenceDate**