Foundation Framework Reference

Core Services Layer: Foundation

2008-06-27

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

Contents

Figures and Tables

The Foundation Framework

Framework	/System/Library/Frameworks/Foundation.framework
Header file directories	/System/Library/Frameworks/Foundation.framework/Headers
Declared in:	FoundationErrors.h NSArray.h NSAutoreleasePool.h NSBundle.h NSByteOrder.h NSCalendar.h NSCharacterSet.h NSCoder.h NSData.h NSDate.h NSDateFormatter.h NSDecimal.h NSDecimalNumber.h NSDictionary.h NSEnumerator.h NSError.h NSException.h NSFileHandle.h NSFileManager.h NSFormatter.h NSHTTPCookie.h NSHTTPCookieStorage.h NSIndexPath.h NSIndexSet.h NSInvocation.h NSKeyValueCoding.h NSKeyValueObserving.h NSKeyedArchiver.h NSLocale.h NSLock.h NSMethodSignature.h NSNetServices.h NSNotification.h NSNotificationQueue.h NSNull.h NSNumberFormatter.h NSObjCRuntime.h

NSObject.h
NSOperation.h
NSPathUtilities.h
NSPort.h
NSProcessInfo.h
NSPropertyList.h
NSProxy.h
NSRange.h
NSRunLoop.h
NSScanner.h
NSSet.h
NSSortDescriptor.h
NSStream.h
NSString.h
NSThread.h
NSTimeZone.h
NSTimer.h
NSURL.h
NSURLAuthenticationChallenge.h
NSURLCache.h
NSURLConnection.h
NSURLCredential.h
NSURLCredentialStorage.h
NSURLError.h
NSURLProtectionSpace.h
NSURLProtocol.h
NSURLRequest.h
NSURLResponse.h
NSUserDefaults.h
NSValue.h
NSXMLParser.h
NSZone.h

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

Introduction

The Foundation framework defines a base layer of Objective-C classes. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Objective-C language. The Foundation framework is designed with these goals in mind:

- Provide a small set of basic utility classes.
- Make software development easier by introducing consistent conventions for things such as deallocation.
- Support Unicode strings, object persistence, and object distribution.
- Provide a level of OS independence, to enhance portability.

The Foundation framework includes the root object class, classes representing basic data types such as strings and byte arrays, collection classes for storing other objects, classes representing system information such as dates, and classes representing communication ports. See Figure I-1 (page 26) for a list of those classes that make up the Foundation framework.

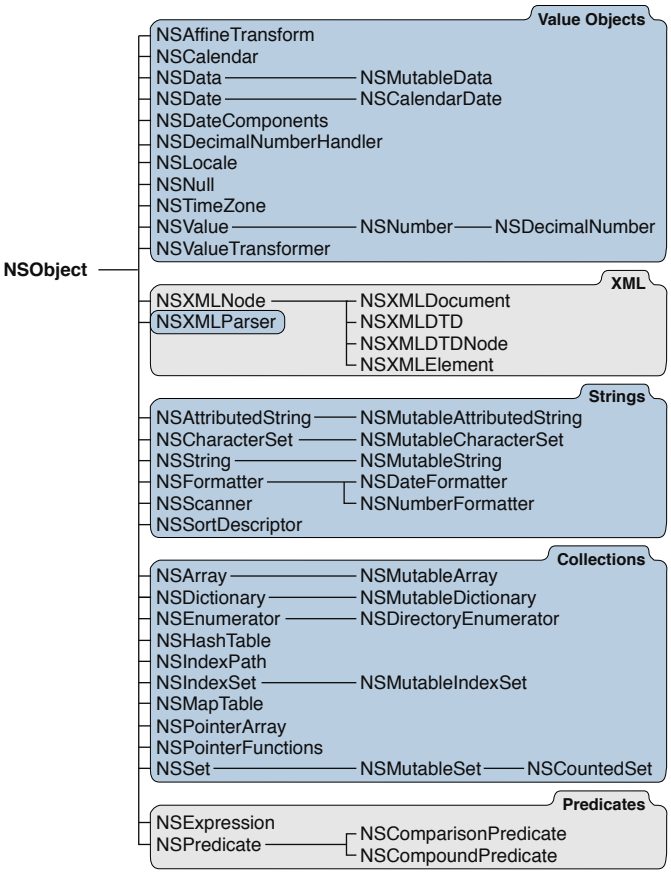
The Foundation framework introduces several paradigms to avoid confusion in common situations, and to introduce a level of consistency across class hierarchies. This consistency is done with some standard policies, such as that for object ownership (that is, who is responsible for disposing of objects), and with abstract classes like `NSEnumerator`. These new paradigms reduce the number of special and exceptional cases in an API and allow you to code more efficiently by reusing the same mechanisms with various kinds of objects.

Foundation Framework Classes

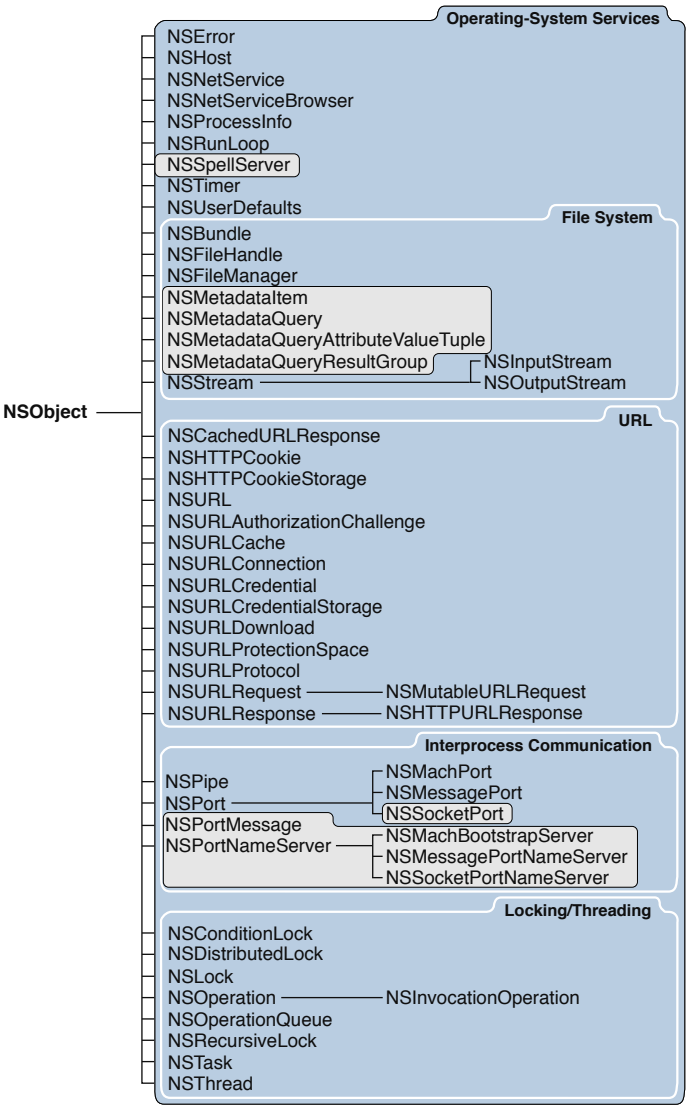
The Foundation class hierarchy is rooted in the Foundation framework's `NSObject` class (see Figure I-1 (page 26)). The remainder of the Foundation framework consists of several related groups of classes as well as a few individual classes. Many of the groups form what are called class clusters—abstract classes that work as umbrella interfaces to a versatile set of private subclasses. `NSString` and `NSMutableString`, for example, act as brokers for instances of various private subclasses optimized for different kinds of storage needs. Depending on the method you use to create a string, an instance of the appropriate optimized class will be returned to you.

Note: In the following class-hierarchy diagrams, blue-shaded areas include classes that are available in Mac OS X and iPhone OS; gray-shaded areas include classes that are available in Mac OS X only.

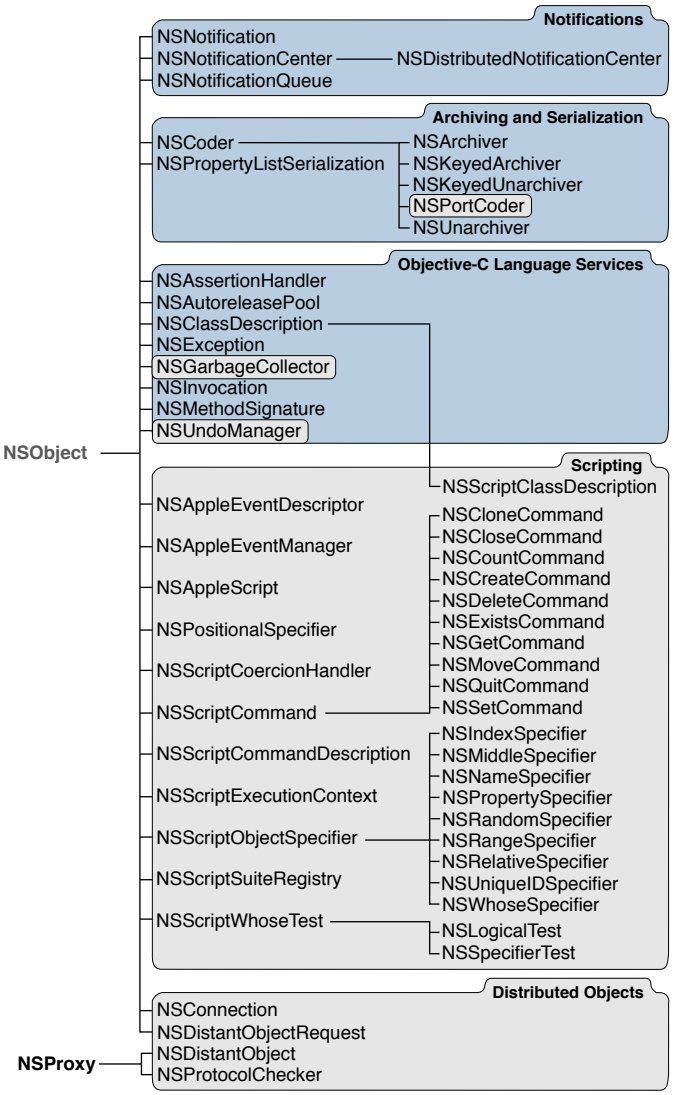
Figure I-1 Cocoa Objective-C Hierarchy for Foundation



Objective-C Foundation Continued



Objective-C Foundation Continued



Many of these classes have closely related functionality:

- Data storage. `NSData` and `NSString` provide object-oriented storage for arrays of bytes. NSValue and NSNumber provide object-oriented storage for arrays of simple C data values. `NSArray`, `NSDictionary`, and `NSSet` provide storage for Objective-C objects of any class.
- Text and strings. `NSCharacterSet` represents various groupings of characters that are used by the `NSString` and `NSScanner` classes. The `NSString` classes represent text strings and provide methods for searching, combining, and comparing strings. An `NSScanner` object is used to scan numbers and words from an `NSString` object.
- Dates and times. The `NSDate`, `NSTimeZone`, and `NSCalendar` classes store times and dates and represent calendrical information. They offer methods for calculating date and time differences. Together with `NSLocale`, they provide methods for displaying dates and times in many formats, and for adjusting times and dates based on location in the world.

- Application coordination and timing. `NSNotification`, `NSNotificationCenter`, and `NSNotificationQueue` provide systems that an object can use to notify all interested observers of changes that occur. You can use an NSTimer object to send a message to another object at specific intervals.
- Object creation and disposal. `NSAutoreleasePool` is used to implement the delayed-release feature of the Foundation framework.
- Object distribution and persistence. The data that an object contains can be represented in an architecture-independent way using `NSPropertyListSerialization`. The `NSCoder` and its subclasses take this process a step further by allowing class information to be stored along with the data. The resulting representations are used for archiving and for object distribution.
- Operating-system services. Several classes are designed to insulate you from the idiosyncrasies of various operating systems. `NSFileManager` provides a consistent interface for file operations (creating, renaming, deleting, and so on). `NSThread` and `NSProcessInfo` let you create multithreaded applications and query the environment in which an application runs.
- URL loading system. A set of classes and protocols provide access to common Internet protocols.

Classes

NSArray Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (`NSObject`)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSArray.h Foundation/NSKeyValueCoding.h Foundation/NSKeyValueObserving.h Foundation/NSPathUtilities.h Foundation/NSPredicate.h Foundation/NSSortDescriptor.h
Companion guides:	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide Property List Programming Guide for Cocoa Predicate Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSArray` and its subclass `NSMutableArray` manage collections of objects called **arrays**. `NSArray` creates static arrays, and `NSMutableArray` creates dynamic arrays.

The `NSArray` and `NSMutableArray` classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert an array of one type to the other.

`NSArray` and `NSMutableArray` are part of a class cluster, so arrays are not actual instances of the `NSArray` or `NSMutableArray` classes but of one of their private subclasses. Although an array's class is private, its interface is public, as declared by these abstract superclasses, `NSArray` and `NSMutableArray`.

`NSArray`'s two primitive methods—`count` (page 47) and `objectAtIndex:` (page 57)—provide the basis for all other methods in its interface. The `count` method returns the number of elements in the array; `objectAtIndex:` gives you access to the array elements by index, with index values starting at `0`.

The methods `objectEnumerator` (page 58) and `reverseObjectEnumerator` (page 60) also grant sequential access to the elements of the array, differing only in the direction of travel through the elements. These methods are provided so that arrays can be traversed in a manner similar to that used for objects of other collection classes, such as `NSDictionary`. See the `objectEnumerator` method description for a code excerpt that shows how to use these methods to access the elements of an array. In Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see `NSFastEnumeration`).

`NSArray` provides methods for querying the elements of the array. The `indexOfObject:` (page 50) method searches the array for the object that matches its argument. To determine whether the search is successful, each element of the array is sent an `isEqual:` (page 1306) message, as declared in the `NSObject` protocol. Another method, `indexOfObjectIdenticalTo:` (page 51), is provided for the less common case of determining whether a specific object is present in the array. The `indexOfObjectIdenticalTo:` method tests each element in the array to see whether its `id` matches that of the argument.

`NSArray`'s `filteredArrayUsingPredicate:` method allows you to create a new array from an existing array filtered using a predicate (see *Predicate Programming Guide*).

`NSArray`'s `makeObjectsPerformSelector:` (page 56) and `makeObjectsPerformSelector:withObject:` (page 57) methods let you send messages to all objects in the array. To act on the array as a whole, a variety of other methods are defined. You can create a sorted version of the array (`sortedArrayUsingSelector:` (page 64) and

sortedArrayUsingFunction:context: (page 62), extract a subset of the array (subarrayWithRange: (page 64)), or concatenate the elements of an array of NSString objects into a single string (componentsJoinedByString: (page 46)). In addition, you can compare two arrays using the isEqualToArray: (page 56) and firstObjectCommonWithArray: (page 49) methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with arrayByAddingObject: (page 45) and arrayByAddingObjectsFromArray: (page 45).

Arrays maintain strong references to their contents—in a managed memory environment, each object receives a retain message before its id is added to the array and a release message when it is removed from the array or when the array is deallocated. If you want a collection with different object ownership semantics, consider using *CFArray Reference*, NSPointerArray, or NSHashTable instead.

NSArray is "toll-free bridged" with its Core Foundation counterpart, *CFArray Reference*. What this means is that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object, providing you cast one type to the other. Therefore, in an API where you see an NSArray * parameter, you can pass in a CFArrayRef, and in an API where you see a CFArrayRef parameter, you can pass in an NSArray instance. This arrangement also applies to your concrete subclasses of NSArray. See *Carbon-Cocoa Integration Guide* for more information on toll-free bridging.

Subclassing Notes

Most developers would not have any reason to subclass NSArray. The class does well what it is designed to do—maintain an ordered collection of objects. But there are situations where a custom NSArray object might come in handy. Here are a few possibilities:

- Changing how NSArray stores the elements of its collection. You might do this for performance reasons or for better compatibility with legacy code.
- Changing how NSArray retains and releases its elements.
- Acquiring more information about what is happening to the collection (for example, statistics gathering).

Methods to Override

Any subclass of NSArray *must* override the primitive instance methods count (page 47) and objectAtIndex: (page 57). These methods must operate on the backing store that you provide for the elements of the collection. For this backing store you can use a static array, a standard NSArray object, or some other data type or mechanism. You may also choose to override, partially or fully, any other NSArray method for which you want to provide an alternative implementation.

You might want to implement an initializer for your subclass that is suited to the backing store that the subclass is managing. The NSArray class does not have a designated initializer, so your initializer need only invoke the init (page 803) method of super. The NSArray class adopts the NSCopying, NSMutableCopying, and NSCoding protocols; if you want instances of your own custom subclass created from copying or coding, override the methods in these protocols.

Remember that NSArray is the public interface for a class cluster and what this entails for your subclass. The primitive methods of NSArray do not include any designated initializers. This means that you must provide the storage for your subclass and implement the primitive methods that directly act on that storage.

Special Considerations

In most cases your custom `NSArray` class should conform to Cocoa's object-ownership conventions. Thus you must send `retain` (page 1312) to each object that you add to your collection and `release` (page 1310) to each object that you remove from the collection. Of course, if the reason for subclassing `NSArray` is to implement object-retention behavior different from the norm (for example, a non-retaining array), then you can ignore this requirement.

Alternatives to Subclassing

Before making a custom class of `NSArray`, investigate `NSPointerArray`, `NSHashTable`, and the corresponding Core Foundation type, *CFArray Reference*. Because `NSArray` and CFArray are "toll-free bridged," you can substitute a CFArray object for a `NSArray` object in your code (with appropriate casting). Although they are corresponding types, CFArray and `NSArray` do not have identical interfaces or implementations, and you can sometimes do things with CFArray that you cannot easily do with `NSArray`. For example, CFArray provides a set of callbacks, some of which are for implementing custom retain-release behavior. If you specify `NULL` implementations for these callbacks, you can easily get a non-retaining array.

If the behavior you want to add supplements that of the existing class, you could write a category on `NSArray`. Keep in mind, however, that this category will be in effect for all instances of `NSArray` that you use, and this might have unintended consequences.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

NSMutableCopying
- `- mutableCopyWithZone:` (page 1300)

Tasks

Creating an Array

`+ array` (page 40)
Creates and returns an empty array.

`+ arrayWithArray:` (page 40)
Creates and returns an array containing the objects in another given array.

`+ arrayWithContentsOfFile:` (page 41)
Creates and returns an array containing the contents of the file specified by a given path.

+ `arrayWithContentsOfURL:` (page 41)
  Creates and returns an array containing the contents specified by a given URL.
+ `arrayWithObject:` (page 42)
  Creates and returns an array containing a given object.
+ `arrayWithObjects:` (page 42)
  Creates and returns an array containing the objects in the argument list.
+ `arrayWithObjects:count:` (page 43)
  Creates and returns an array that includes a given number of objects from a given C array.

Initializing an Array

- `initWithArray:` (page 52)
  Initializes a newly allocated array by placing in it the objects contained in a given array.
- `initWithArray:copyItems:` (page 53)
  Initializes a newly allocated array using *anArray* as the source of data objects for the array.
- `initWithContentsOfFile:` (page 53)
  Initializes a newly allocated array with the contents of the file specified by a given path.
- `initWithContentsOfURL:` (page 54)
  Initializes a newly allocated array with the contents of the location specified by a given URL.
- `initWithObjects:` (page 54)
  Initializes a newly allocated array by placing in it the objects in the argument list.
- `initWithObjects:count:` (page 55)
  Initializes a newly allocated array to include a given number of objects from a given C array.

Querying an Array

- `containsObject:` (page 46)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `count` (page 47)
  Returns the number of objects currently in the receiver.
- `getObjects:` (page 49)
  Copies all the objects contained in the receiver to *aBuffer*.
- `getObjects:range:` (page 50)
  Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.
- `indexOfObject:` (page 50)
  Returns the lowest index whose corresponding array value is equal to a given object.
- `indexOfObject:inRange:` (page 50)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectIdenticalTo:` (page 51)
  Returns the lowest index whose corresponding array value is identical to a given object.
- `indexOfObjectIdenticalTo:inRange:` (page 52)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .

- `lastObject` (page 56)
  Returns the object in the array with the highest index value.
- `objectAtIndex:` (page 57)
  Returns the object located at *index*.
- `objectsAtIndexes:` (page 58)
  Returns an array containing the objects in the receiver at the indexes specified by a given index set.
- `objectEnumerator` (page 58)
  Returns an enumerator object that lets you access each object in the receiver.
- `reverseObjectEnumerator` (page 60)
  Returns an enumerator object that lets you access each object in the receiver, in reverse order.

Sending Messages to Elements

- `makeObjectsPerformSelector:` (page 56)
  Sends to each object in the receiver the message identified by a given selector, starting with the first object and continuing through the array to the last object.
- `makeObjectsPerformSelector:withObject:` (page 57)
  Sends the *aSelector* message to each object in the array, starting with the first object and continuing through the array to the last object.

Comparing Arrays

- `firstObjectCommonWithArray:` (page 49)
  Returns the first object contained in the receiver that's equal to an object in another given array.
- `isEqualToArray:` (page 56)
  Compares the receiving array to another array.

Deriving New Arrays

- `arrayByAddingObject:` (page 45)
  Returns a new array that is a copy of the receiver with a given object added to the end.
- `arrayByAddingObjectsFromArray:` (page 45)
  Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.
- `subarrayWithRange:` (page 64)
  Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

Sorting

- `sortedArrayHint` (page 61)
  Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 63).

- `sortedArrayUsingFunction:context:` (page 62)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.
- `sortedArrayUsingFunction:context:hint:` (page 63)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.
- `sortedArrayUsingDescriptors:` (page 61)
  Returns a copy of the receiver sorted as specified by a given array of sort descriptors.
- `sortedArrayUsingSelector:` (page 64)
  Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

Working with String Elements

- `componentsJoinedByString:` (page 46)
  Constructs and returns an `NSString` object that is the result of interposing a given separator between the elements of the receiver's array.

Creating a Description

- `description` (page 47)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:` (page 48)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 48)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `writeToFile:atomically:` (page 65)
  Writes the contents of the receiver to a file at a given path.
- `writeToURL:atomically:` (page 66)
  Writes the contents of the receiver to the location specified by a given URL.

Collecting Paths

- `pathsMatchingExtensions:` (page 59)
  Returns an array containing all the pathname elements in the receiver that have filename extensions from a given array.

Key-Value Observing

- `addObserver:forKeyPath:options:context:` (page 44)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 59)
  Raises an exception.

- addObserver:toObjectsAtIndexes:forKeyPath:options:context: (page 44)
  Registers *anObserver* to receive key value observer notifications for the specified *keypath* relative to the objects at *indexes*.
- removeObserver:fromObjectsAtIndexes:forKeyPath: (page 60)
  Removes *anObserver* from all key value observer notifications associated with the specified *keyPath* relative to the receiver's objects at *indexes*.

Key-Value Coding

- setValue:forKey: (page 61)
  Invokes setValue:forKey: on each of the receiver's items using the specified *value* and *key*.
- valueForKey: (page 65)
  Returns an array containing the results of invoking valueForKey: using *key* on each of the receiver's objects.

Class Methods

array

Creates and returns an empty array.

```
+ (id)array
```

Return Value
An empty array.

Discussion
This method is used by mutable subclasses of NSArray.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ arrayWithObject: (page 42)
+ arrayWithObjects: (page 42)

Declared In
NSArray.h

arrayWithArray:

Creates and returns an array containing the objects in another given array.

```
+ (id)arrayWithArray:(NSArray *)anArray
```

Parameters
anArray
  An array.

Return Value
An array containing the objects in *anArray*.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ arrayWithObjects: (page 42)
- initWithObjects: (page 54)

Declared In
NSArray.h

arrayWithContentsOfFile:
Creates and returns an array containing the contents of the file specified by a given path.

```
+ (id)arrayWithContentsOfFile:(NSString *)aPath
```

Parameters
aPath

The path to a file containing a string representation of an array produced by the writeToFile:atomically: (page 65) method.

Return Value
An array containing the contents of the file specified by *aPath*. Returns nil if the file can't be opened or if the contents of the file can't be parsed into an array.

Discussion
The array representation in the file identified by *aPath* must contain only property list objects (NSString, NSData, NSArray, or NSDictionary objects).

Availability
Available in iPhone OS 2.0 and later.

See Also
- writeToFile:atomically: (page 65)

Declared In
NSArray.h

arrayWithContentsOfURL:
Creates and returns an array containing the contents specified by a given URL.

```
+ (id)arrayWithContentsOfURL:(NSURL *)aURL
```

Parameters
aURL

The location of a file containing a string representation of an array produced by the writeToURL:atomically: (page 66) method.

Return Value
An array containing the contents specified by *aURL*. Returns `nil` if the location can't be opened or if the contents of the location can't be parsed into an array.

Discussion
The array representation at the location identified by *aURL* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- writeToURL:atomically:` (page 66)

Declared In
`NSArray.h`

arrayWithObject:

Creates and returns an array containing a given object.

```
+ (id)arrayWithObject:(id)anObject
```

Parameters
anObject
An object.

Return Value
An array containing the single element *anObject*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ array` (page 40)
`+ arrayWithObjects:` (page 42)

Declared In
`NSArray.h`

arrayWithObjects:

Creates and returns an array containing the objects in the argument list.

```
+ (id)arrayWithObjects:(id)firstObj, ...
```

Parameters
firstObj, ...
A comma-separated list of objects ending with `nil`.

Return Value
An array containing the objects in the argument list.

Discussion
This code example creates an array containing three different types of element:

```
NSArray *myArray;
NSDate *aDate = [NSDate distantFuture];
NSValue *aValue = [NSNumber numberWithInt:5];
NSString *aString = @"a string";

myArray = [NSArray arrayWithObjects:aDate, aValue, aString, nil];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
+ array (page 40)
+ arrayWithObject: (page 42)

Declared In
NSArray.h

arrayWithObjects:count:
Creates and returns an array that includes a given number of objects from a given C array.

```
+ (id)arrayWithObjects:(const id *)objects count:(NSUInteger)count
```

Parameters
objects
A C array of objects.

count
The number of values from the *objects* C array to include in the new array. This number will be the count of the new array—it must not be negative or greater than the number of elements in *objects*.

Return Value
A new array including the first *count* objects from *objects*.

Discussion
Elements are added to the new array in the same order they appear in *objects*, up to but not including index *count*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- getObjects: (page 49)
- getObjects:range: (page 50)

Declared In
NSArray.h

Instance Methods

addObserver:forKeyPath:options:context:

Raises an exception.

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options context:(void *)context
```

Parameters

observer

The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 1287).

keyPath

The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options

A combination of the `NSKeyValueObservingOptions` (page 1291) values that specifies what is included in observation notifications. For possible values, see `NSKeyValueObservingOptions`.

context

Arbitrary data that is passed to *observer* in `observeValueForKeyPath:ofObject:change:context:` (page 1287).

Special Considerations

`NSArray` objects are not observable, so this method raises an exception when invoked on an `NSArray` object. Instead of observing an array, observe the to-many relationship for which the array is the collection of related objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `removeObserver:forKeyPath:` (page 59)
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 44)

Declared In

`NSKeyValueObserving.h`

addObserver:toObjectsAtIndexes:forKeyPath:options:context:

Registers *anObserver* to receive key value observer notifications for the specified *keypath* relative to the objects at *indexes*.

```
- (void)addObserver:(NSObject *)anObserver toObjectsAtIndexes:(NSIndexSet *)indexes
    forKeyPath:(NSString *)keyPath options:(NSKeyValueObservingOptions)options
    context:(void *)context
```

Discussion

The *options* determine what is included in the notifications, and the *context* is passed in the notifications.

This is not merely a convenience method; invoking this method is potentially much faster than repeatedly invoking `addObserver:forKeyPath:options:context:` (page 1284).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 60)

Declared In
`NSKeyValueObserving.h`

arrayByAddingObject:

Returns a new array that is a copy of the receiver with a given object added to the end.

```
- (NSArray *)arrayByAddingObject:(id)anObject
```

Parameters

anObject
An object.

Return Value
A new array that is a copy of the receiver with *anObject* added to the end.

Discussion
If *anObject* is `nil`, an `NSInvalidArgumentException` is raised.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- addObject:` (page 571) (`NSMutableArray`)

Declared In
`NSArray.h`

arrayByAddingObjectsFromArray:

Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.

```
- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)otherArray
```

Parameters

otherArray
An array.

Return Value
A new array that is a copy of the receiver with the objects contained in *otherArray* added to the end.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addObjectsFromArray: (page 571) (NSMutableArray)

Declared In
NSArray.h

componentsJoinedByString:

Constructs and returns an NSString object that is the result of interposing a given separator between the elements of the receiver's array.

```
- (NSString *)componentsJoinedByString:(NSString *)separator
```

Parameters
separator
The string to interpose between the elements of the receiver's array.

Return Value
An NSString object that is the result of interposing *separator* between the elements of the receiver's array. If the receiver has no elements, returns an NSString object representing an empty string.

Discussion
For example, this code excerpt writes "here be dragons" to the console:

```
NSArray *pathArray = [NSArray arrayWithObjects:@"here",
    @"be", @"dragons", nil];
NSLog(@"%@",
    [pathArray componentsJoinedByString:@" "]);
```

Special Considerations
Each element in the receiver's array must handle description.

Availability
Available in iPhone OS 2.0 and later.

See Also
- componentsSeparatedByString: (page 985) (NSString)

Declared In
NSArray.h

containsObject:

Returns a Boolean value that indicates whether a given object is present in the receiver.

```
- (BOOL)containsObject:(id)anObject
```

Parameters
anObject
An object.

Return Value
YES if *anObject* is present in the receiver, otherwise NO.

Discussion
This method determines whether *anObject* is present in the receiver by sending an `isEqual:` (page 1306) message to each of the receiver's objects (and passing *anObject* as the parameter to each `isEqual:` message).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `indexOfObject:` (page 50)
- `indexOfObjectIdenticalTo:` (page 51)

Declared In
`NSArray.h`

count

Returns the number of objects currently in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of objects currently in the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `objectAtIndex:` (page 57)

Declared In
`NSArray.h`

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `descriptionWithLocale:` (page 48)
- `descriptionWithLocale:indent:` (page 48)

Declared In
`NSArray.h`

descriptionWithLocale:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters

locale

An `NSLocale` object or an `NSDictionary` object that specifies options used for formatting each of the receiver's elements (where recognized). Specify `nil` if you don't want the elements formatted.

Return Value

A string that represents the contents of the receiver, formatted as a property list.

Discussion

For a description of how *locale* is applied to each element in the receiving array, see `descriptionWithLocale:indent:` (page 48).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- description` (page 47)
- `- descriptionWithLocale:indent:` (page 48)

Declared In

`NSArray.h`

descriptionWithLocale:indent:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale indent:(NSUInteger)level
```

Parameters

locale

An `NSLocale` object or an `NSDictionary` object that specifies options used for formatting each of the receiver's elements (where recognized). Specify `nil` if you don't want the elements formatted.

level

A level of indent, to make the output more readable: set *level* to `0` to use four spaces to indent, or `1` to indent the output with a tab character.

Return Value

A string that represents the contents of the receiver, formatted as a property list.

Discussion

The returned `NSString` object contains the string representations of each of the receiver's elements, in order, from first to last. To obtain the string representation of a given element, `descriptionWithLocale:indent:` proceeds as follows:

- If the element is an `NSString` object, it is used as is.

- If the element responds to `descriptionWithLocale:indent:`, that method is invoked to obtain the element's string representation.
- If the element responds to `descriptionWithLocale:` (page 48), that method is invoked to obtain the element's string representation.
- If none of the above conditions is met, the element's string representation is obtained by invoking its `description` (page 47) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- description` (page 47)
- `- descriptionWithLocale:` (page 48)

Declared In
`NSArray.h`

firstObjectCommonWithArray:
Returns the first object contained in the receiver that's equal to an object in another given array.

```
- (id)firstObjectCommonWithArray:(NSArray *)otherArray
```

Parameters
otherArray
An array.

Return Value
Returns the first object contained in the receiver that's equal to an object in *otherArray*. If no such object is found, returns `nil`.

Discussion
This method uses `isEqual:` (page 1306) to check for object equality.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- containsObject:` (page 46)

Declared In
`NSArray.h`

getObjects:
Copies all the objects contained in the receiver to *aBuffer*.

```
- (void)getObjects:(id *)aBuffer
```

Availability
Available in iPhone OS 2.0 and later.

See Also
+ arrayWithObjects:count: (page 43)

Declared In
NSArray.h

getObjects:range:

Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.

```
- (void)getObjects:(id *)aBuffer range:(NSRange)aRange
```

Availability
Available in iPhone OS 2.0 and later.

See Also
+ arrayWithObjects:count: (page 43)

Declared In
NSArray.h

indexOfObject:

Returns the lowest index whose corresponding array value is equal to a given object.

```
- (NSUInteger)indexOfObject:(id)anObject
```

Parameters
anObject
An object.

Return Value
The lowest index whose corresponding array value is equal to *anObject*. If none of the objects in the receiver is equal to *anObject*, returns NSNotFound.

Discussion
Objects are considered equal if isEqual: (page 1306) returns YES.

Availability
Available in iPhone OS 2.0 and later.

See Also
- containsObject: (page 46)
- indexOfObjectIdenticalTo: (page 51)

Declared In
NSArray.h

indexOfObject:inRange:

Returns the lowest index within a specified range whose corresponding array value is equal to a given object .

```
- (NSUInteger)indexOfObject:(id)anObject inRange:(NSRange)range
```

Parameters

anObject

An object.

range

The range of indexes in the receiver within which to search for *anObject*.

Return Value

The lowest index within *range* whose corresponding array value is equal to *anObject*. If none of the objects within *range* is equal to *anObject*, returns `NSNotFound`.

Discussion

Objects are considered equal if `isEqual:` (page 1306) returns `YES`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `containsObject:` (page 46)
- `indexOfObjectIdenticalTo:inRange:` (page 52)

Declared In

`NSArray.h`

indexOfObjectIdenticalTo:

Returns the lowest index whose corresponding array value is identical to a given object.

```
- (NSUInteger)indexOfObjectIdenticalTo:(id)anObject
```

Parameters

anObject

An object.

Return Value

The lowest index whose corresponding array value is identical to *anObject*. If none of the objects in the receiver is identical to *anObject*, returns `NSNotFound`.

Discussion

Objects are considered identical if their object addresses are the same.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `containsObject:` (page 46)
- `indexOfObject:` (page 50)

Declared In

`NSArray.h`

indexOfObjectIdenticalTo:inRange:

Returns the lowest index within a specified range whose corresponding array value is equal to a given object .

```
- (NSUInteger)indexOfObjectIdenticalTo:(id)anObject inRange:(NSRange)range
```

Parameters

anObject

An object.

range

The range of indexes in the receiver within which to search for *anObject*.

Return Value

The lowest index within *range* whose corresponding array value is identical to *anObject*. If none of the objects within *range* is identical to *anObject*, returns `NSNotFound`.

Discussion

Objects are considered identical if their object addresses are the same.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- containsObject:` (page 46)
- `- indexOfObject:inRange:` (page 50)

Declared In

`NSArray.h`

initWithArray:

Initializes a newly allocated array by placing in it the objects contained in a given array.

```
- (id)initWithArray:(NSArray *)anArray
```

Parameters

anArray

An array.

Return Value

An array initialized to contain the objects in *anArray*. The returned object might be different than the original receiver.

Discussion

After an immutable array has been initialized in this way, it cannot be modified.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `+ arrayWithObject:` (page 42)
- `- initWithObjects:` (page 54)

Declared In
NSArray.h

initWithArray:copyItems:

Initializes a newly allocated array using *anArray* as the source of data objects for the array.

```
- (id)initWithArray:(NSArray *)array copyItems:(BOOL)flag
```

Parameters

array

An array.

flag

If YES, each object in *array* receives a copyWithZone: message to create a copy of the object. In a managed memory environment, this is instead of the retain message the object would otherwise receive. The object copy is then added to the returned array.

If NO, then in a managed memory environment each object in *array* simply receives a retain message as it's added to the returned array.

Return Value
An array initialized to contain the objects—or if *flag* is YES, copies of the objects—in *array*. The returned object might be different than the original receiver.

Discussion
After an immutable array has been initialized in this way, it cannot be modified.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithArray: (page 52)
+ arrayWithObject: (page 42)
- initWithObjects: (page 54)

Declared In
NSArray.h

initWithContentsOfFile:

Initializes a newly allocated array with the contents of the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)aPath
```

Parameters

aPath

The path to a file containing a string representation of an array produced by the writeToFile:atomically: (page 65) method.

Return Value
An array initialized to contain the contents of the file specified by *aPath* or nil if the file can't be opened or the contents of the file can't be parsed into an array. The returned object might be different than the original receiver.

Discussion
The array representation in the file identified by *aPath* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ arrayWithContentsOfFile:` (page 41)
`- writeToFile:atomically:` (page 65)

Declared In
`NSArray.h`

initWithContentsOfURL:
Initializes a newly allocated array with the contents of the location specified by a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters
aURL
The location of a file containing a string representation of an array produced by the `writeToURL:atomically:` (page 66) method.

Return Value
An array initialized to contain the contents specified by *aURL*. Returns `nil` if the location can't be opened or if the contents of the location can't be parsed into an array. The returned object might be different than the original receiver.

Discussion
The array representation at the location identified by *aURL* must contain only property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ arrayWithContentsOfURL:` (page 41)
`- writeToURL:atomically:` (page 66)

Declared In
`NSArray.h`

initWithObjects:
Initializes a newly allocated array by placing in it the objects in the argument list.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters
firstObj, ...
A comma-separated list of objects ending with `nil`.

Return Value
An array initialized to include the objects in the argument list. The returned object might be different than the original receiver.

Discussion
After an immutable array has been initialized in this way, it can't be modified.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithObjects:count:` (page 55)
+ `arrayWithObjects:` (page 42)
- `initWithArray:` (page 52)

Declared In
`NSArray.h`

initWithObjects:count:
Initializes a newly allocated array to include a given number of objects from a given C array.

```
- (id)initWithObjects:(const id *)objects
    count:(NSUInteger)count
```

Parameters
objects
A C array of objects.

count
The number of values from the *objects* C array to include in the new array. This number will be the count of the new array—it must not be negative or greater than the number of elements in *objects*.

Return Value
A newly allocated array including the first *count* objects from *objects*. The returned object might be different than the original receiver.

Discussion
Elements are added to the new array in the same order they appear in *objects*, up to but not including index *count*.

After an immutable array has been initialized in this way, it can't be modified.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithObjects:` (page 54)
+ `arrayWithObjects:` (page 42)
- `initWithArray:` (page 52)

Declared In
`NSArray.h`

isEqualToArray:

Compares the receiving array to another array.

```
- (BOOL)isEqualToArray:(NSArray *)otherArray
```

Parameters

otherArray
An array.

Return Value

YES if the contents of *otherArray* are equal to the contents of the receiver, otherwise NO.

Discussion

Two arrays have equal contents if they each hold the same number of objects and objects at a given index in each array satisfy the isEqual: (page 1306) test.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSArray.h

lastObject

Returns the object in the array with the highest index value.

```
- (id)lastObject
```

Return Value

The object in the array with the highest index value. If the array is empty, returns nil.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeLastObject (page 575) (NSMutableArray)

Declared In

NSArray.h

makeObjectsPerformSelector:

Sends to each object in the receiver the message identified by a given selector, starting with the first object and continuing through the array to the last object.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector
```

Parameters

aSelector
A selector that identifies the message to send to the objects in the receiver. The method must not take any arguments, and must not have the side effect of modifying the receiving array.

Discussion

This method raises an NSInvalidArgumentException if *aSelector* is NULL.

Availability
Available in iPhone OS 2.0 and later.

See Also
- makeObjectsPerformSelector:withObject: (page 57)

Declared In
NSArray.h

makeObjectsPerformSelector:withObject:

Sends the *aSelector* message to each object in the array, starting with the first object and continuing through the array to the last object.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector
A selector that identifies the message to send to the objects in the receiver. The method must take a single argument of type id, and must not have the side effect of modifying the receiving array.

anObject
The object to send as the argument to each invocation of the *aSelector* method.

Discussion
This method raises an NSInvalidArgumentException if *aSelector* is NULL.

Availability
Available in iPhone OS 2.0 and later.

See Also
- makeObjectsPerformSelector: (page 56)

Declared In
NSArray.h

objectAtIndex:

Returns the object located at *index*.

```
- (id)objectAtIndex:(NSUInteger)index
```

Parameters

index
An index within the bounds of the receiver.

Return Value
The object located at *index*.

Discussion
If *index* is beyond the end of the array (that is, if *index* is greater than or equal to the value returned by count), an NSRangeException is raised.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `count` (page 47)
- `objectsAtIndexes:` (page 58)

Declared In
`NSArray.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver, in order, from the element at the lowest index upwards.

Discussion
Returns an enumerator object that lets you access each object in the receiver, in order, starting with the element at index 0, as in:

```
NSEnumerator *enumerator = [myArray objectEnumerator];
id anObject;

while (anObject = [enumerator nextObject]) {
    /* code to act on each element as it is returned */
}
```

Special Considerations
When you use this method with mutable subclasses of `NSArray`, you must not modify the array during enumeration.

On Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see `NSFastEnumeration`).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `reverseObjectEnumerator` (page 60)
- `nextObject` (page 341) (`NSEnumerator`)

Declared In
`NSArray.h`

objectsAtIndexes:

Returns an array containing the objects in the receiver at the indexes specified by a given index set.

```
- (NSArray *)objectsAtIndexes:(NSIndexSet *)indexes
```

Return Value
An array containing the objects in the receiver at the indexes specified by *indexes*.

Discussion
The returned objects are in the ascending order of their indexes in *indexes*, so that object in returned array with higher index in indexes will follow the object with smaller index in *indexes*.

Raises an `NSRangeException` exception if any location in *indexes* exceeds the bounds of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `count` (page 47)
- `objectAtIndex:` (page 57)

Declared In
`NSArray.h`

pathsMatchingExtensions:
Returns an array containing all the pathname elements in the receiver that have filename extensions from a given array.

```
- (NSArray *)pathsMatchingExtensions:(NSArray *)filterTypes
```

Parameters
filterTypes
: An array of `NSString` objects containing filename extensions. The extensions should not include the dot (".") character.

Return Value
An array containing all the pathname elements in the receiver that have filename extensions from the *filterTypes* array.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSPathUtilities.h`

removeObserver:forKeyPath:
Raises an exception.

```
- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
```

Parameters
observer
: The object to remove as an observer.

keyPath
: A key-path, relative to the receiver, for which *observer* is registered to receive KVO change notifications. This value must not be `nil`.

Special Considerations

`NSArray` objects are not observable, so this method raises an exception when invoked on an `NSArray` object. Instead of observing an array, observe the to-many relationship for which the array is the collection of related objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addObserver:forKeyPath:options:context:` (page 44)
- `removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 60)

Declared In

`NSKeyValueObserving.h`

removeObserver:fromObjectsAtIndexes:forKeyPath:

Removes *anObserver* from all key value observer notifications associated with the specified *keyPath* relative to the receiver's objects at *indexes*.

```
- (void)removeObserver:(NSObject *)anObserver fromObjectsAtIndexes:(NSIndexSet
    *)indexes forKeyPath:(NSString *)keyPath
```

Discussion

This is not merely a convenience method; invoking this method is potentially much faster than repeatedly invoking `removeObserver:forKeyPath:` (page 1288).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 44)

Declared In

`NSKeyValueObserving.h`

reverseObjectEnumerator

Returns an enumerator object that lets you access each object in the receiver, in reverse order.

```
- (NSEnumerator *)reverseObjectEnumerator
```

Return Value

An enumerator object that lets you access each object in the receiver, in order, from the element at the highest index down to the element at index `0`.

Special Considerations

When you use this method with mutable subclasses of `NSArray`, you must not modify the array during enumeration.

On Mac OS X v10.5 and later, it is more efficient to use the fast enumeration protocol (see `NSFastEnumeration`).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `objectEnumerator` (page 58)
- `nextObject` (page 341) (`NSEnumerator`)

Declared In
`NSArray.h`

setValue:forKey:

Invokes `setValue:forKey:` on each of the receiver's items using the specified *`value`* and *`key`*.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `valueForKey:` (page 65)

Declared In
`NSKeyValueCoding.h`

sortedArrayHint

Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 63).

```
- (NSData *)sortedArrayHint
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sortedArrayUsingFunction:context:hint:` (page 63)

Declared In
`NSArray.h`

sortedArrayUsingDescriptors:

Returns a copy of the receiver sorted as specified by a given array of sort descriptors.

```
- (NSArray *)sortedArrayUsingDescriptors:(NSArray *)sortDescriptors
```

Parameters
`sortDescriptors`
An array of `NSSortDescriptor` objects.

Return Value
A copy of the receiver sorted as specified by *sortDescriptors*.

Discussion
The first descriptor specifies the primary key path to be used in sorting the receiver's contents. Any subsequent descriptors are used to further refine sorting of objects with duplicate values. See `NSSortDescriptor` for additional information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- sortedArrayUsingSelector:` (page 64)
- `- sortedArrayUsingFunction:context:` (page 62)
- `- sortedArrayUsingFunction:context:hint:` (page 63)

Declared In
`NSSortDescriptor.h`

sortedArrayUsingFunction:context:
Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.

```
- (NSArray *)sortedArrayUsingFunction:(NSInteger (*)(id, id, void *))comparator
    context:(void *)context
```

Discussion
The new array contains references to the receiver's elements, not copies of them.

The comparison function is used to compare two elements at a time and should return `NSOrderedAscending` if the first element is smaller than the second, `NSOrderedDescending` if the first element is larger than the second, and `NSOrderedSame` if the elements are equal. Each time the comparison function is called, it's passed *context* as its third argument. This allows the comparison to be based on some outside parameter, such as whether character sorting is case-sensitive or case-insensitive.

Given *anArray* (an array of `NSNumber` objects) and a comparison function of this type:

```
NSInteger intSort(id num1, id num2, void *context)
{
    int v1 = [num1 intValue];
    int v2 = [num2 intValue];
    if (v1 < v2)
        return NSOrderedAscending;
    else if (v1 > v2)
        return NSOrderedDescending;
    else
        return NSOrderedSame;
}
```

A sorted version of *anArray* is created in this way:

```
NSArray *sortedArray; sortedArray = [anArray sortedArrayUsingFunction:intSort
context:NULL];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sortedArrayUsingDescriptors:` (page 61)
- `sortedArrayUsingFunction:context:hint:` (page 63)
- `sortedArrayUsingSelector:` (page 64)

Declared In
`NSArray.h`

sortedArrayUsingFunction:context:hint:

Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *comparator*.

```
- (NSArray *)sortedArrayUsingFunction:(NSInteger (*)(id, id, void *))comparator
    context:(void *)context hint:(NSData *)hint
```

Discussion
The new array contains references to the receiver's elements, not copies of them.

This method is similar to `sortedArrayUsingFunction:context:` (page 62), except that it uses the supplied hint to speed the sorting process. When you know the array is nearly sorted, this method is faster than `sortedArrayUsingFunction:context:`. If you sorted a large array (`N` entries) once, and you don't change it much (`P` additions and deletions, where `P` is much smaller than `N`), then you can reuse the work you did in the original sort by conceptually doing a merge sort between the `N` "old" items and the `P` "new" items.

To obtain an appropriate hint, use `sortedArrayHint` (page 61). You should obtain this hint when the original array has been sorted, and keep hold of it until you need it, after the array has been modified. The hint is computed by `sortedArrayHint` (page 61) in `O(N)` (where `N` is the number of items). This assumes that items in the array implement a `-hash` method. Given a suitable hint, and assuming that the hash function is a "good" hash function, `-sortedArrayUsingFunction:context:hint:` (page 63) sorts the array in `O(P*LOG(P)+N)` where `P` is the number of adds or deletes. This is an improvement over the unhinted sort, `O(N*LOG(N))`, when `P` is small.

The hint is simply an array of size `N` containing the `N` hashes. To re-sort you need internally to create a map table mapping a hash to the index. Using this map table on the new array, you can get a first guess for the indices, and then sort that. For example, a sorted array {A, B, D, E, F} with corresponding hash values {25, 96, 78, 32, 17}, may be subject to small changes that result in contents {E, A, C, B, F}. The mapping table maps the hashes {25, 96, 78, 32, 17} to the indices {#0, #1, #2, #3, #4}. If the hashes for {E, A, C, B, F} are {32, 25, 99, 96, 17}, then by using the mapping table you can get a first order sort {#3, #0, ?, #1, #4}, so therefore create an initial semi-sorted array {A, B, E, F}, and then perform a cheap merge sort with {C} that yields {A, B, C, E, F}.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sortedArrayUsingDescriptors:` (page 61)
- `sortedArrayUsingFunction:context:` (page 62)

- sortedArrayUsingSelector: (page 64)

Declared In
NSArray.h

sortedArrayUsingSelector:

Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

```
- (NSArray *)sortedArrayUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that identifies the method to use to compare two elements at a time. The method should return NSOrderedAscending if the receiver is smaller than the argument, NSOrderedDescending if the receiver is larger than the argument, and NSOrderedSame if they are equal.

Return Value
An array that lists the receiver's elements in ascending order, as determined by the comparison method specified by the selector *comparator*.

Discussion
The new array contains references to the receiver's elements, not copies of them.

The *comparator* message is sent to each object in the array and has as its single argument another object in the array.

For example, an array of NSString objects can be sorted by using the caseInsensitiveCompare: (page 979) method declared in the NSString class. Assuming *anArray* exists, a sorted version of the array can be created in this way:

```
NSArray *sortedArray =
    [anArray sortedArrayUsingSelector:@selector(caseInsensitiveCompare:)];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- sortedArrayUsingDescriptors: (page 61)
- sortedArrayUsingFunction:context: (page 62)
- sortedArrayUsingFunction:context:hint: (page 63)

Declared In
NSArray.h

subarrayWithRange:

Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

```
- (NSArray *)subarrayWithRange:(NSRange)range
```

Parameters

range

A range within the receiver's range of elements.

Return Value

A new array containing the receiver's elements that fall within the limits specified by *range*.

Discussion

If *range* isn't within the receiver's range of elements, an `NSRangeException` is raised.

For example, the following code example creates an array containing the elements found in the first half of *wholeArray* (assuming *wholeArray* exists).

```
NSArray *halfArray;
NSRange theRange;

theRange.location = 0;
theRange.length = [wholeArray count] / 2;

halfArray = [wholeArray subarrayWithRange:theRange];
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSArray.h`

valueForKey:

Returns an array containing the results of invoking `valueForKey:` using *key* on each of the receiver's objects.

```
- (id)valueForKey:(NSString *)key
```

Discussion

The returned array contains `NSNull` elements for each object that returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setValue:forKey:` (page 61)

Declared In

`NSKeyValueCoding.h`

writeToFile:atomically:

Writes the contents of the receiver to a file at a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters

path

The path at which to write the contents of the receiver.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1031) before invoking this method.

flag

If `YES`, the array is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If `NO`, the array is written directly to *path*. The `YES` option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value

`YES` if the file is written successfully, otherwise `NO`.

Discussion

If the receiver's contents are all property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects), the file written by this method can be used to initialize a new array with the class method `arrayWithContentsOfFile:` (page 41) or the instance method `initWithContentsOfFile:` (page 53). This method recursively validates that all the contained objects are property list objects before writing out the file, and returns `NO` if all the objects are not property list objects, since the resultant file would not be a valid property list.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithContentsOfFile:` (page 53)

Declared In

`NSArray.h`

writeToURL:atomically:

Writes the contents of the receiver to the location specified by a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

Parameters

aURL

The location at which to write the receiver.

flag

If `YES`, the array is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If `NO`, the array is written directly to *aURL*. The `YES` option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value

`YES` if the location is written successfully, otherwise `NO`.

Discussion

If the receiver's contents are all property list objects (`NSString`, `NSData`, `NSArray`, or `NSDictionary` objects), the location written by this method can be used to initialize a new array with the class method `arrayWithContentsOfURL:` (page 41) or the instance method `initWithContentsOfURL:` (page 54).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithContentsOfURL:` (page 54)

Declared In
`NSArray.h`

NSAssertionHandler Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSException.h
Companion guide:	Assertions and Logging

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSAssertionHandler` objects are automatically created to handle false assertions. Assertion macros, such as `NSAssert` and `NSCAssert`, are used to evaluate a condition, and, if the condition evaluates to false, the macros pass a string to an `NSAssertionHandler` object describing the failure. Each thread has its own `NSAssertionHandler` object. When invoked, an assertion handler prints an error message that includes the method and class (or function) containing the assertion and raises an `NSInternalInconsistencyException`.

You create assertions only using the assertion macros—you rarely need to invoke `NSAssertionHandler` methods directly. The macros for use inside methods and functions send `handleFailureInMethod:object:file:lineNumber:description:` (page 71) and `handleFailureInFunction:file:lineNumber:description:` (page 71) messages respectively to

the current assertion handler. The assertion handler for the current thread is obtained using the `currentHandler` (page 70) class method. If you need to customize the behavior of `NSAssertionHandler`, create a subclass, overriding the above two methods, and install your instance into the current thread's attributes dictionary with the key `NSAssertionHandler`.

Tasks

Handling Assertion Failures

`+ currentHandler` (page 70)
Returns the `NSAssertionHandler` object associated with the current thread.

`- handleFailureInFunction:file:lineNumber:description:` (page 71)
Logs (using `NSLog`) an error message that includes the name of the function, the name of the file, and the line number.

`- handleFailureInMethod:object:file:lineNumber:description:` (page 71)
Logs (using `NSLog`) an error message that includes the name of the method that failed, the class name of the object, the name of the source file, and the line number.

Class Methods

currentHandler

Returns the `NSAssertionHandler` object associated with the current thread.

```
+ (NSAssertionHandler *)currentHandler
```

Return Value

The `NSAssertionHandler` object associated with the current thread.

Discussion

If no assertion handler is associated with the current thread, this method creates one and assigns it to the thread.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSException.h`

Instance Methods

handleFailureInFunction:file:lineNumber:description:

Logs (using `NSLog`) an error message that includes the name of the function, the name of the file, and the line number.

```
- (void)handleFailureInFunction:(NSString *)functionName file:(NSString *)fileName
    lineNumber:(NSInteger)line description:(NSString *)format, ...
```

Parameters

functionName
The function that failed.

object
The object that failed.

fileName
The name of the source file.

line
The line in which the failure occurred.

format,...
A format string followed by a comma-separated list of arguments to substitute into the format string. See Formatting String Objects for more information.

Discussion

Raises `NSInternalInconsistencyException`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSException.h`

handleFailureInMethod:object:file:lineNumber:description:

Logs (using `NSLog`) an error message that includes the name of the method that failed, the class name of the object, the name of the source file, and the line number.

```
- (void)handleFailureInMethod:(SEL)selector object:(id)object file:(NSString
    *)fileName lineNumber:(NSInteger)line description:(NSString *)format, ...
```

Parameters

selector
The selector for the method that failed

object
The object that failed.

fileName
The name of the source file.

line
The line in which the failure occurred.

format,...

A format string followed by a comma-separated list of arguments to substitute into the format string. See Formatting String Objects for more information.

Discussion

Raises `NSInternalInconsistencyException`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSException.h`

NSAutoreleasePool Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSAutoreleasePool.h
Companion guide:	Memory Management Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSAutoreleasePool` class is used to support Cocoa's memory management system. An autorelease pool stores objects that are sent a `release` message when the pool itself is released.

In a managed memory environment (as opposed to one which uses garbage collection) an `NSAutoreleasePool` object contains objects that have received an `autorelease` (page 1303) message and when deallocated sends a `release` (page 1310) message to each of those objects. An object can be put into the same pool several times and receives a `release` (page 1310) message for each time it was put into the pool. Thus, sending `autorelease` (page 1303) instead of `release` (page 1310) to an object extends the lifetime of that object at least until the pool itself is released (it may be longer if the object is subsequently retained).

Cocoa expects there to be an autorelease pool always available. If a pool is not available, autoreleased objects do not get released and you leak memory. `NSAutoreleasePool` objects are automatically created and destroyed in the main thread of applications based on the Application Kit, so your code normally does not have to deal with them. The Application Kit creates a pool at the beginning of the event loop and releases it at the end, thereby periodically releasing any autoreleased objects generated while processing events.

If your application creates a lot of temporary autoreleased objects within the event loop, it may be beneficial to create local autorelease pools to help to minimize the peak memory footprint. You create an `NSAutoreleasePool` object with the usual `alloc` and `init` messages and dispose of it with `release` (page 77) or `drain` (page 76) (to understand the difference, see "Garbage Collection" (page 74)). You should always drain an autorelease pool in the same context (invocation of a method or function, or body of a loop) that it was created. See Autorelease Pools for more details and several code samples using autorelease pools.

Each thread (including the main thread) maintains its own stack of `NSAutoreleasePool` objects (see "Threads" (page 74)). As new pools are created, they get added to the top of the stack. When pools are deallocated, they are removed from the stack. Autoreleased objects are placed into the top autorelease pool for the current thread. When a thread terminates, it automatically releases all of the autorelease pools associated with itself.

Threads

If you are making Cocoa calls outside of the Application Kit's main thread—for example if you create a Foundation-only application or if you detach a thread—you need to create your own autorelease pool.

If your application or thread is long-lived and potentially generates a lot of autoreleased objects, you should periodically destroy and create autorelease pools (like the Application Kit does on the main thread); otherwise, autoreleased objects accumulate and your memory footprint grows. If, however, your detached thread does not make Cocoa calls, you do not need to create an autorelease pool.

Note: If you are creating secondary threads using the POSIX thread APIs instead of `NSThread` objects, you cannot use Cocoa, including `NSAutoreleasePool`, unless Cocoa is in multithreading mode. Cocoa enters multithreading mode only after detaching its first `NSThread` object. To use Cocoa on secondary POSIX threads, your application must first detach at least one `NSThread` object, which can immediately exit. You can test whether Cocoa is in multithreading mode with the `NSThread` class method `isMultiThreaded` (page 1054).

Garbage Collection

In a garbage collected environment, there is no need for autorelease pools. You can, though, use autorelease pools to hint to the collector that collection may be appropriate. In a garbage collected environment, sending a `drain` (page 76) message to a pool triggers garbage collection if necessary (`release` (page 77), however, is a no-op). In a managed memory environment, `drain` (page 76) has the same effect as `release` (page 77). Typically, therefore, if you write code that is intended to be compiled for garbage collected and managed memory environments, you should use `drain` (page 76) instead of `release` (page 77).

Tasks

Managing a Pool

- `release` (page 77)
  Releases and pops the receiver.
- `drain` (page 76)
  In a garbage collected environment, triggers garbage collection if memory allocated since last collection is greater than the current threshold; otherwise behaves as `release` (page 1310).
- `autorelease` (page 76)
  Raises an exception.
- `retain` (page 77)
  Raises an exception.

Adding an Object to a Pool

+ `addObject:` (page 75)
  Adds a given object to the active autorelease pool in the current thread.
- `addObject:` (page 76)
  Adds a given object to the receiver

Class Methods

addObject:

Adds a given object to the active autorelease pool in the current thread.

```
+ (void)addObject:(id)object
```

Parameters

object
  The object to add to the active autorelease pool in the current thread.

Discussion

The same object may be added several times to the active pool and, when the pool is deallocated, it will receive a `release` (page 1310) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 1303) to *object* instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addObject:` (page 76)

Declared In
NSAutoreleasePool.h

Instance Methods

addObject:

Adds a given object to the receiver

```
- (void)addObject:(id)object
```

Parameters

object
The object to add to the receiver.

Discussion
The same object may be added several times to the same pool and, when the pool is deallocated, it will receive a `release` (page 1310) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 1303) to *object* instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ addObject:` (page 75)

Declared In
NSAutoreleasePool.h

autorelease

Raises an exception.

```
- (id)autorelease
```

Return Value
`self`

Discussion
In a managed memory environment, this method raises an exception.

drain

In a garbage collected environment, triggers garbage collection if memory allocated since last collection is greater than the current threshold; otherwise behaves as `release` (page 1310).

```
- (void)drain
```

Discussion
In a garbage collected environment, this method triggers garbage collection if the memory allocated since the last collection is greater than the current threshold. This method ultimately calls `objc_collect_if_needed`.

In a managed memory environment, this behaves the same as calling `release` (page 1310).

Special Considerations
In a garbage collected environment, `release` is a no-op, so unless you do not want to give the compiler a hint it is important to use `drain` in any code that may be compiled for a garbage collected environment.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSAutoreleasePool.h`

release
Releases and pops the receiver.

```
- (void)release
```

Discussion
When an autorelease pool is deallocated, it sends a release message to all its autoreleased objects. If an object is added several times to the same pool, when the pool is deallocated it receives a `release` (page 1310) message for each time it was added.

Special Considerations
In a garbage collected environment, `release` is a no-op, so typically you should use `drain` (page 76) in any code that may be compiled for a garbage collected environment.

retain
Raises an exception.

```
- (id)retain
```

Return Value
`self`

Discussion
In a managed memory environment, this method raises an exception.

NSBundle Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSBundle.h
Companion guides:	Bundle Programming Guide Resource Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSBundle` object represents a location in the file system that groups code and resources that can be used in a program. `NSBundle` objects locate program resources, dynamically load and unload executable code, and assist in localization. You build a bundle in Xcode using one of these project types: Application, Framework, Loadable Bundle, Palette.

See also NSBundle Additions in the Application Kit framework, which defines methods for loading nib files and locating image resources.

Unlike some other Foundation classes with corresponding Core Foundation names (such as `NSString` and CFString), `NSBundle` objects cannot be cast ("toll-free bridged") to CFBundle references. If you need functionality provided in CFBundle, you can still create a CFBundle and use the CFBundle API. See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Initializing an NSBundle

`- initWithPath:` (page 92)
Returns an `NSBundle` object initialized to correspond to a given directory.

Getting an NSBundle

`+ bundleForClass:` (page 83)
Returns the `NSBundle` object with which a given class is associated.

`+ bundleWithIdentifier:` (page 84)
Returns the previously created `NSBundle` instance that has a given bundle identifier.

`+ bundleWithPath:` (page 84)
Returns an `NSBundle` object that corresponds to the specified directory.

`+ mainBundle` (page 85)
Returns the `NSBundle` object that corresponds to the directory where the current application executable is located.

`+ allBundles` (page 83)
Returns an array of all the application's non-framework bundles.

`+ allFrameworks` (page 83)
Returns an array of all of the application's bundles that represent frameworks.

Getting a Bundled Class

`- classNamed:` (page 89)
Returns the `Class` object for the specified name.

`- principalClass` (page 102)
Returns the receiver's principal class.

Finding a Resource

`+ pathForResource:ofType:inDirectory:` (page 85)
Returns the full pathname for the resource file identified by a given name and extension and residing in a given bundle directory.

`- pathForResource:ofType:` (page 97)
Returns the full pathname for the resource identified by a given name and specified file extension.

- pathForResource:ofType:inDirectory: (page 98)
  Returns the full pathname for the resource identified by the given name and file extension and located in the specified bundle subdirectory.
- pathForResource:ofType:inDirectory:forLocalization: (page 99)
  Returns the full pathname for the resource identified by the given name and file extension, located in the specified bundle subdirectory, and limited to global resources and those associated with the specified localization.
+ pathsForResourcesOfType:inDirectory: (page 86)
  Returns an array containing the pathnames for all bundle resources having a given extension and residing in the bundle directory specified by a given path.
- pathsForResourcesOfType:inDirectory: (page 99)
  Returns an array containing the pathnames for all bundle resources having the specified filename extension and residing in the resource subdirectory.
- pathsForResourcesOfType:inDirectory:forLocalization: (page 100)
  Returns an array containing the pathnames for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, and limited to global resources and those associated with the specified localization.
- resourcePath (page 103)
  Returns the full pathname of the receiving bundle's subdirectory containing resources.

Getting the Bundle Directory

- bundlePath (page 89)
  Returns the full pathname of the receiver's bundle directory.

Getting Bundle Information

- builtInPlugInsPath (page 88)
  Returns the full pathname of the receiver's subdirectory containing plug-ins.
- bundleIdentifier (page 89)
  Returns the receiver's bundle identifier.
- executablePath (page 91)
  Returns the full pathname of the receiver's executable file.
- infoDictionary (page 91)
  Returns a dictionary that contains information about the receiver.
- objectForInfoDictionaryKey: (page 96)
  Returns the value associated with a given key in the receiver's property list.
- pathForAuxiliaryExecutable: (page 96)
  Returns the full pathname of the executable with a given name in the receiver's bundle.
- privateFrameworksPath (page 103)
  Returns the full pathname of the receiver's subdirectory containing private frameworks.
- sharedFrameworksPath (page 104)
  Returns the full pathname of the receiver's subdirectory containing shared frameworks.
- sharedSupportPath (page 104)
  Returns the full pathname of the receiver's subdirectory containing shared support files.

Managing Localized Resources

- `localizedStringForKey:value:table:` (page 95)
  Returns a localized version of the string designated by a given key in a given table.

Loading a Bundle's Code

- `executableArchitectures` (page 90)
  Returns an array of numbers indicating the architecture types supported by the bundle's executable.
- `preflightAndReturnError:` (page 101)
  Returns a Boolean value indicating whether the bundle's executable code could be loaded successfully.
- `load` (page 93)
  Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.
- `loadAndReturnError:` (page 93)
  Loads the bundle's executable code and returns any errors.
- `isLoaded` (page 92)
  Obtains information about the load status of a bundle.
- `unload` (page 104)
  Unloads the code associated with the receiver.

Managing Localizations

+ `preferredLocalizationsFromArray:` (page 87)
  Returns one or more localizations from the specified list that a bundle object would use to locate resources for the current user.
+ `preferredLocalizationsFromArray:forPreferences:` (page 88)
  Returns the localizations that a bundle object would prefer, given the specified bundle and user preference localizations.
- `localizations` (page 94)
  Returns a list of all the localizations contained within the receiver's bundle.
- `developmentLocalization` (page 90)
  Returns the localization used to create the bundle.
- `preferredLocalizations` (page 101)
  Returns one or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.
- `localizedInfoDictionary` (page 94)
  Returns a dictionary with the keys from the bundle's localized property list.

Class Methods

allBundles

Returns an array of all the application's non-framework bundles.

```
+ (NSArray *)allBundles
```

Return Value

An array of all the application's non-framework bundles.

Discussion

The returned array includes the main bundle and all bundles that have been dynamically created but doesn't contain any bundles that represent frameworks.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSBundle.h`

allFrameworks

Returns an array of all of the application's bundles that represent frameworks.

```
+ (NSArray *)allFrameworks
```

Return Value

An array of all of the application's bundles that represent frameworks. Only frameworks with one or more Objective-C classes in them are included.

Discussion

The returned array includes frameworks that are linked into an application when the application is built and bundles for frameworks that have been dynamically created.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSBundle.h`

bundleForClass:

Returns the `NSBundle` object with which a given class is associated.

```
+ (NSBundle *)bundleForClass:(Class)aClass
```

Parameters

aClass

A class.

Return Value
The `NSBundle` object that dynamically loaded *`aClass`* (a loadable bundle), the `NSBundle` object for the framework in which *`aClass`* is defined, or the main bundle object if *`aClass`* was not dynamically loaded or is not defined in a framework.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ mainBundle` (page 85)
`+ bundleWithPath:` (page 84)

Declared In
`NSBundle.h`

bundleWithIdentifier:

Returns the previously created `NSBundle` instance that has a given bundle identifier.

```
+ (NSBundle *)bundleWithIdentifier:(NSString *)identifier
```

Parameters
`identifier`
The identifier for an existing `NSBundle` instance.

Return Value
The previously created `NSBundle` instance that has the bundle identifier *`identifier`*. Returns `nil` if the requested bundle is not found.

Discussion
This method is typically used by frameworks and plug-ins to locate their own bundle at runtime. This method may be somewhat more efficient than trying to locate the bundle using the `bundleForClass:` (page 83) method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

bundleWithPath:

Returns an `NSBundle` object that corresponds to the specified directory.

```
+ (NSBundle *)bundleWithPath:(NSString *)fullPath
```

Parameters
`fullPath`
The path to a directory. This must be a full pathname for a directory; if it contains any symbolic links, they must be resolvable.

Return Value
The `NSBundle` object that corresponds to *`fullPath`*, or `nil` if *`fullPath`* does not identify an accessible bundle directory.

Discussion
This method allocates and initializes the returned object if there is no existing `NSBundle` associated with *`fullPath`*, in which case it returns the existing object.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ mainBundle` (page 85)
`+ bundleForClass:` (page 83)
`- initWithPath:` (page 92)

Declared In
`NSBundle.h`

mainBundle
Returns the `NSBundle` object that corresponds to the directory where the current application executable is located.

```
+ (NSBundle *)mainBundle
```

Return Value
The `NSBundle` object that corresponds to the directory where the application executable is located, or `nil` if a bundle object could not be created.

Discussion
This method allocates and initializes a bundle object if one doesn't already exist. The new object corresponds to the directory where the application executable is located. Be sure to check the return value to make sure you have a valid bundle. This method may return a valid bundle object even for unbundled applications.

In general, the main bundle corresponds to an application file package or application wrapper: a directory that bears the name of the application and is marked by a "`.app`" extension.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ bundleForClass:` (page 83)
`+ bundleWithPath:` (page 84)

Declared In
`NSBundle.h`

pathForResource:ofType:inDirectory:
Returns the full pathname for the resource file identified by a given name and extension and residing in a given bundle directory.

```
+ (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)bundlePath
```

Parameters

name

The name of a resource file contained in the bundle specified by *bundlePath*.

extension

If *extension* is an empty string or `nil`, the returned pathname is the first one encountered that exactly matches *name*.

bundlePath

The path of a top-level bundle directory. This must be a valid path. For example, to specify the bundle directory for an application, you might specify the path `/Applications/MyApp.app`.

Return Value

The full pathname for the resource file or `nil` if the file could not be located. This method also returns `nil` if the bundle specified by the `bundlePath` parameter does not exist or is not a readable directory.

Discussion

The method first looks for a matching resource file in the nonlocalized resource directory (typically `Resources`) of the specified bundle. If a matching resource file is not found, it then looks in the top level of any available language-specific "`.lproj`" directories. (The search order for the language-specific directories corresponds to the user's preferences.) It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

Note: This method is best suited only for the occasional retrieval of resource files. In most cases where you need to retrieve bundle resources, it is preferable to use the `NSBundle` instance methods instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `localizedStringForKey:value:table:` (page 95)
- `pathForResource:ofType:` (page 97)
- `pathForResource:ofType:inDirectory:` (page 98)
+ `pathsForResourcesOfType:inDirectory:` (page 86)
- `pathsForResourcesOfType:inDirectory:` (page 99)

Declared In

`NSBundle.h`

pathsForResourcesOfType:inDirectory:

Returns an array containing the pathnames for all bundle resources having a given extension and residing in the bundle directory specified by a given path.

```
+ (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)bundlePath
```

Parameters

extension

If *extension* is an empty string or `nil`, all bundle resources in the top-level resource directories are returned.

bundlePath
The top-level directory of a bundle. This must represent a valid path.

Return Value
An array containing the full pathnames for all bundle resources with the specified extension. This method returns an empty array of no matching resource files are found. It also returns an empty array if the bundle specified by the `bundlePath` parameter does not exist or is not a readable directory.

Discussion
This method provides a means for dynamically discovering multiple bundle resources of the same type.

The method first looks for matching resource files in the nonlocalized resource directory (typically `Resources`) of the specified bundle. It then looks in the top level of any available language-specific "`.lproj`" directories. It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

Note: This method is best suited only for the occasional retrieval of resource files. In most cases where you need to retrieve bundle resources, it is preferable to use the `NSBundle` instance methods instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- localizedStringForKey:value:table:` (page 95)
- `- pathForResource:ofType:` (page 97)
- `- pathForResource:ofType:inDirectory:` (page 98)
- `+ pathForResource:ofType:inDirectory:` (page 85)

Declared In
`NSBundle.h`

preferredLocalizationsFromArray:

Returns one or more localizations from the specified list that a bundle object would use to locate resources for the current user.

```
+ (NSArray *)preferredLocalizationsFromArray:(NSArray *)localizationsArray
```

Parameters
localizationsArray
An array of `NSString` objects, each of which specifies the name of a localization that the bundle supports.

Return Value
An array of `NSString` objects containing the preferred localizations. These strings are ordered in the array according to the current user's language preferences and are taken from the strings in the *localizationsArray* parameter.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

preferredLocalizationsFromArray:forPreferences:

Returns the localizations that a bundle object would prefer, given the specified bundle and user preference localizations.

```
+ (NSArray *)preferredLocalizationsFromArray:(NSArray *)localizationsArray
    forPreferences:(NSArray *)preferencesArray
```

Parameters

localizationsArray
An array of NSString objects, each of which specifies the name of a localization that the bundle supports.

preferencesArray
An array of NSString objects containing the user's preferred localizations. If this parameter is nil, the method uses the current user's localization preferences.

Return Value
An array of NSString objects containing the preferred localizations. These strings are ordered in the array according to the specified preferences and are taken from the strings in the *localizationsArray* parameter.

Discussion
Use the argument *localizationsArray* to specify the supported localizations of the bundle and use *preferencesArray* to specify the user's localization preferences.

If none of the user-preferred localizations are available in the bundle, this method chooses one of the bundle localizations and returns it.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

Instance Methods

builtInPlugInsPath

Returns the full pathname of the receiver's subdirectory containing plug-ins.

```
- (NSString *)builtInPlugInsPath
```

Return Value
The full pathname of the receiving bundle's subdirectory containing plug-ins.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

bundleIdentifier

Returns the receiver's bundle identifier.

```
- (NSString *)bundleIdentifier
```

Return Value
The receiver's bundle identifier, which is defined by the `CFBundleIdentifier` key in the bundle's information property list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- infoDictionary (page 91)

Declared In
NSBundle.h

bundlePath

Returns the full pathname of the receiver's bundle directory.

```
- (NSString *)bundlePath
```

Return Value
The full pathname of the receiver's bundle directory.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

classNamed:

Returns the `Class` object for the specified name.

```
- (Class)classNamed:(NSString *)className
```

Parameters
className
The name of a class.

Return Value
The `Class` object for *className*. Returns `NIL` if *className* is not one of the classes associated with the receiver or if there is an error loading the executable code containing the class implementation.

Discussion

If the bundle's executable code is not yet loaded, this method dynamically loads it into memory. Classes (and categories) are loaded from just one file within the bundle directory; this code file has the same name as the directory, but without the extension ("`.bundle`", "`.app`", "`.framework`"). As a side effect of code loading, the receiver posts `NSBundleDidLoadNotification` (page 106) after all classes and categories have been loaded; see "Notifications" (page 106) for details.

The following example loads a bundle's executable code containing the class "`FaxWatcher`":

```
- (void)loadBundle:(id)sender
{
    Class exampleClass;
    id newInstance;
    NSString *str = @"~/BundleExamples/BundleExample.bundle";
    NSBundle *bundleToLoad = [NSBundle bundleWithPath:str];
    if (exampleClass = [bundleToLoad classNamed:@"FaxWatcher"])  {
        newInstance = [[exampleClass alloc] init];
    // [newInstance doSomething];
    }
}
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- `principalClass` (page 102)
- `load` (page 93)

Declared In

`NSBundle.h`

developmentLocalization

Returns the localization used to create the bundle.

```
- (NSString *)developmentLocalization
```

Return Value

The localization used to create the bundle.

Discussion

The returned localization corresponds to the value in the `CFBundleDevelopmentRegion` key of the bundle's property list (`Info.plist`).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSBundle.h`

executableArchitectures

Returns an array of numbers indicating the architecture types supported by the bundle's executable.

```
- (NSArray *)executableArchitectures
```

Return Value
An array of `NSNumber` objects, each of which contains an integer value corresponding to a supported processor architecture. For a list of common architecture types, see the constants in "Mach-O Architecture" (page 105). If the bundle does not contain a Mach-O executable, this method returns `nil`.

Discussion
This method scans the bundle's Mach-O executable and returns all of the architecture types it finds. Because they are taken directly from the executable, the returned values may not always correspond to one of the well-known CPU types defined in "Mach-O Architecture" (page 105).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

executablePath

Returns the full pathname of the receiver's executable file.

```
- (NSString *)executablePath
```

Return Value
The full pathname of the receiving bundle's executable file.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

infoDictionary

Returns a dictionary that contains information about the receiver.

```
- (NSDictionary *)infoDictionary
```

Return Value
A dictionary, constructed from the bundle's `Info.plist` file, that contains information about the receiver. If the bundle does not contain an `Info.plist` file, a valid dictionary is returned but this dictionary contains only private keys that are used internally by the `NSBundle` class.

Discussion
Common keys for accessing the values of the dictionary are `CFBundleIdentifier`, `NSMainNibFile`, and `NSPrincipalClass`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- principalClass` (page 102)

Declared In
NSBundle.h

initWithPath:

Returns an NSBundle object initialized to correspond to a given directory.

```
- (id)initWithPath:(NSString *)fullPath
```

Parameters

fullPath

The path to a directory. This must be a full pathname for a directory; if it contains any symbolic links, they must be resolvable.

Return Value
An NSBundle object initialized to correspond to *fullPath*. This method initializes and returns a new instance only if there is no existing bundle associated with *fullPath*, otherwise it deallocates self and returns the existing object. If *fullPath* doesn't exist or the user doesn't have access to it, returns nil.

Discussion
It's not necessary to allocate and initialize an instance for the main bundle; use the mainBundle (page 85) class method to get this instance. You can also use the bundleWithPath: (page 84) class method to obtain a bundle identified by its directory path.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ bundleForClass: (page 83)

Declared In
NSBundle.h

isLoaded

Obtains information about the load status of a bundle.

```
- (BOOL)isLoaded
```

Return Value
YES if the bundle's code is currently loaded, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- load (page 93)

Declared In
NSBundle.h

load

Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.

```
- (BOOL)load
```

Return Value

`YES` if the method successfully loads the bundle's code or if the code has already been loaded, otherwise `NO`.

Discussion

You can use this method to load the code associated with a dynamically loaded bundle, such as a plug-in or framework. Prior to Mac OS X version 10.5, a bundle would attempt to load its code—if it had any—only once. Once loaded, you could not unload that code. In Mac OS X version 10.5 and later, you can unload a bundle's executable code using the `unload` (page 104) method.

You don't need to load a bundle's executable code to search the bundle's resources.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `loadAndReturnError:` (page 93)
- `isLoaded` (page 92)
- `unload` (page 104)
- `classNamed:` (page 89)
- `principalClass` (page 102)

Declared In

`NSBundle.h`

loadAndReturnError:

Loads the bundle's executable code and returns any errors.

```
- (BOOL)loadAndReturnError:(NSError **)error
```

Parameters

error

On input, a pointer to an error object variable. On output, this variable may contain an error object indicating why the bundle's executable could not be loaded. If no error occurred, this parameter is left unmodified. You may specify `nil` for this parameter if you are not interested in the error information.

Return Value

`YES` if the bundle's executable code was loaded successfully or was already loaded; otherwise, `NO` if the code could not be loaded.

Discussion

If this method returns `NO` and you pass a value for the *error* parameter, a suitable error object is returned in that parameter. Potential errors returned are in the Cocoa error domain and include the types that follow. For a full list of error types, see `FoundationErrors.h`.

- `NSFileNoSuchFileError` - returned if the bundle's executable file was not located.

- `NSExecutableNotLoadableError` - returned if the bundle's executable file exists but could not be loaded. This error is returned if the executable is not recognized as a loadable executable. It can also be returned if the executable is a PEF/CFM executable but the current process does not support that type of executable.
- `NSExecutableArchitectureMismatchError` - returned if the bundle executable does not include code that matches the processor architecture of the current processor.
- `NSExecutableRuntimeMismatchError` - returned if the bundle's required Objective-C runtime information is not compatible with the runtime of the current process.
- `NSExecutableLoadError` - returned if the bundle's executable failed to load for some detectable reason prior to linking. This error might occur if the bundle depends on a framework or library that is missing or if the required framework or library is not compatible with the current architecture or runtime version.
- `NSExecutableLinkError` - returned if the executable failed to load due to link errors but is otherwise alright.

The error object may contain additional debugging information in its description that you can use to identify the cause of the error. (This debugging information should not be displayed to the user.) You can obtain the debugging information by invoking the error object's `description` method in your code or by using the `print-object` command on the error object in gdb.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `load` (page 93)
- `unload` (page 104)

Declared In
`NSBundle.h`

localizations

Returns a list of all the localizations contained within the receiver's bundle.

```
- (NSArray *)localizations
```

Return Value
An array, containing `NSString` objects, that specifies all the localizations contained within the receiver's bundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

localizedInfoDictionary

Returns a dictionary with the keys from the bundle's localized property list.

```
- (NSDictionary *)localizedInfoDictionary
```

Return Value
A dictionary with the keys from the bundle's localized property list (`InfoPList.strings`).

Discussion
This method uses the preferred localization for the current user when determining which resources to return. If the preferred localization is not available, this method chooses the most appropriate localization found in the bundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

localizedStringForKey:value:table:
Returns a localized version of the string designated by a given key in a given table.

```
- (NSString *)localizedStringForKey:(NSString *)key value:(NSString *)value
    table:(NSString *)tableName
```

Parameters
key

The key for a string in the table identified by *tableName*.

value

The value to return if *key* is `nil` or if a localized string for *key* can't be found in the table.

tableName

The receiver's string table to search. If *tableName* is `nil` or is an empty string, the method attempts to use the table in `Localizable.strings`.

Return Value
A localized version of the string designated by *key* in table *tableName*. If *value* is `nil` or an empty string, and a localized string is not found in the table, returns *key*. If *key* and *value* are both `nil`, returns the empty string.

Discussion
For more details about string localization and the specification of a `.strings` file, see "Working With Localized Strings."

Using the user default `NSShowNonLocalizedStrings`, you can alter the behavior of `localizedStringForKey:value:table:` (page 95) to log a message when the method can't find a localized string. If you set this default to `YES` (in the global domain or in the application's domain), then when the method can't find a localized string in the table, it logs a message to the console and capitalizes *key* before returning it.

The following example cycles through a static array of keys when a button is clicked, gets the value for each key from a strings table named `Buttons.strings`, and sets the button title with the returned value:

```
- (void)changeTitle:(id)sender
{
    static int keyIndex = 0;
```

```
    NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];

    NSString *locString = [thisBundle
        localizedStringForKey:assortedKeys[keyIndex++]
        value:@"No translation" table:@"Buttons"];
    [sender setTitle:locString];
    if (keyIndex == MAXSTRINGS) keyIndex=0;
}
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- pathForResource:ofType:` (page 97)
- `- pathForResource:ofType:inDirectory:` (page 98)
- `- pathsForResourcesOfType:inDirectory:` (page 99)
- `+ pathForResource:ofType:inDirectory:` (page 85)
- `+ pathsForResourcesOfType:inDirectory:` (page 86)

Declared In
`NSBundle.h`

objectForInfoDictionaryKey:

Returns the value associated with a given key in the receiver's property list.

```
- (id)objectForInfoDictionaryKey:(NSString *)key
```

Parameters
key

A key in the receiver's property list.

Return Value
The value associated with *key* in the receiver's property list (`Info.plist`). The localized value of a key is returned when one is available.

Discussion
Use of this method is preferred over other access methods because it returns the localized value of a key when one is available.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

pathForAuxiliaryExecutable:

Returns the full pathname of the executable with a given name in the receiver's bundle.

```
- (NSString *)pathForAuxiliaryExecutable:(NSString *)executableName
```

Parameters

executableName

The name of an executable file.

Return Value

The full pathname of the executable *executableName* in the receiver's bundle.

Discussion

This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSBundle.h

pathForResource:ofType:

Returns the full pathname for the resource identified by a given name and specified file extension.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
```

Parameters

name

The name of the resource file.

extension

The file extension of a resource with the name *name*.

Return Value

The full pathname for the resource file or nil if the file could not be located.

Discussion

If *extension* is an empty string or nil, the returned pathname is the first one encountered where the file name exactly matches *name*.

The method first looks for a matching resource file in the nonlocalized resource directory (typically Resources) of the specified bundle. If a matching resource file is not found, it then looks in the top level of any available language-specific ".lproj" directories. (The search order for the language-specific directories corresponds to the user's preferences.) It does not recurse through other subdirectories at any of these locations. For more details see Bundles and Localization.

The following code fragment gets the path to a localized sound, creates an NSSound instance from it, and plays the sound.

```
NSString *soundPath;
NSSound *thisSound;
NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];
if (soundPath = [thisBundle pathForResource:@"Hello" ofType:@"snd"])  {
    thisSound = [[[NSSound alloc] initFromSoundfile:soundPath] autorelease];
    [thisSound play];
}
```

Availability

Available in iPhone OS 2.0 and later.

See Also
- localizedStringForKey:value:table: (page 95)
- pathForResource:ofType: (page 97)
- pathForResource:ofType:inDirectory: (page 98)
+ pathForResource:ofType:inDirectory: (page 85)
+ pathsForResourcesOfType:inDirectory: (page 86)

Declared In
NSBundle.h

pathForResource:ofType:inDirectory:

Returns the full pathname for the resource identified by the given name and file extension and located in the specified bundle subdirectory.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)subpath
```

Parameters
name
The name of the resource file.

extension
The file extension of the specified resource file.

subpath
The name of the bundle subdirectory.

Return Value
The full pathname for the resource file or nil if the file could not be located.

Discussion
If *extension* is an empty string or nil, the returned pathname is the first one encountered where the file name exactly matches *name*.

If *subpath* is nil, this method searches the top-level nonlocalized resource directory (typically Resources) and the top-level of any language-specific directories. For example, suppose you have a modern bundle and specify @"Documentation" for the *subpath* parameter. This method would first look in the Contents/Resources/Documentation directory of the bundle, followed by the Documentation subdirectories of each language-specific .lproj directory. (The search order for the language-specific directories corresponds to the user's preferences.) This method does not recurse through any other subdirectories at any of these locations. For more details see Bundles and Localization.

Availability
Available in iPhone OS 2.0 and later.

See Also
- localizedStringForKey:value:table: (page 95)
- pathForResource:ofType: (page 97)
- pathsForResourcesOfType:inDirectory: (page 99)
+ pathForResource:ofType:inDirectory: (page 85)
+ pathsForResourcesOfType:inDirectory: (page 86)

Declared In
NSBundle.h

pathForResource:ofType:inDirectory:forLocalization:

Returns the full pathname for the resource identified by the given name and file extension, located in the specified bundle subdirectory, and limited to global resources and those associated with the specified localization.

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)extension
    inDirectory:(NSString *)subpath forLocalization:(NSString *)localizationName
```

Parameters

name
The name of the resource file.

extension
The file extension of the specified resource file.

subpath
The name of the bundle subdirectory to search.

localizationName
The name of the localization. This parameter should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension.

Return Value
The full pathname for the resource file or `nil` if the file could not be located.

Discussion
This method is equivalent to `pathForResource:ofType:inDirectory:` (page 98), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

There should typically be little reason to use this method—see Getting the Current Language and Locale. See also preferredLocalizationsFromArray:forPreferences: (page 88) for how to determine what localizations are available.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

pathsForResourcesOfType:inDirectory:

Returns an array containing the pathnames for all bundle resources having the specified filename extension and residing in the resource subdirectory.

```
- (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)subpath
```

Parameters

extension
The file extension of the files to retrieve.

subpath
The name of the bundle subdirectory to search.

Return Value
An array containing the full pathnames for all bundle resources matching the specified criteria. This method returns an empty array of no matching resource files are found.

Discussion
This method provides a means for dynamically discovering multiple bundle resources of the same type. If *extension* is an empty string or `nil`, all bundle resources in the specified resource directory are returned.

The argument *subpath* specifies the name of a specific subdirectory to search within the current bundle's resource directory hierarchy. If *subpath* is `nil`, this method searches the top-level nonlocalized resource directory (typically `Resources`) and the top-level of any language-specific directories. For example, suppose you have a modern bundle and specify `@"Documentation"` for the *subpath* parameter. This method would first look in the `Contents/Resources/Documentation` directory of the bundle, followed by the `Documentation` subdirectories of each language-specific `.lproj` directory. (The search order for the language-specific directories corresponds to the user's preferences.) This method does not recurse through any other subdirectories at any of these locations. For more details see Bundles and Localization.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- localizedStringForKey:value:table:` (page 95)
- `- pathForResource:ofType:` (page 97)
- `- pathForResource:ofType:inDirectory:` (page 98)
- `+ pathForResource:ofType:inDirectory:` (page 85)
- `+ pathsForResourcesOfType:inDirectory:` (page 86)

Declared In
`NSBundle.h`

pathsForResourcesOfType:inDirectory:forLocalization:

Returns an array containing the pathnames for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, and limited to global resources and those associated with the specified localization.

```
- (NSArray *)pathsForResourcesOfType:(NSString *)extension inDirectory:(NSString
    *)subpath forLocalization:(NSString *)localizationName
```

Parameters

extension
The file extension of the files to retrieve.

subpath
The name of the bundle subdirectory to search.

localizationName
The name of the localization. This parameter should correspond to the name of one of the bundle's language-specific resource directories without the `.lproj` extension.

Return Value
An array containing the full pathnames for all bundle resources matching the specified criteria. This method returns an empty array of no matching resource files are found.

Discussion
This method is equivalent to `pathsForResourcesOfType:inDirectory:` (page 99), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

preferredLocalizations

Returns one or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.

```
- (NSArray *)preferredLocalizations
```

Return Value
One or more localizations contained in the receiver's bundle that the receiver uses to locate resources based on the user's preferences.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ preferredLocalizationsFromArray:` (page 87)
`- localizations` (page 94)

Declared In
`NSBundle.h`

preflightAndReturnError:

Returns a Boolean value indicating whether the bundle's executable code could be loaded successfully.

```
- (BOOL)preflightAndReturnError:(NSError **)error
```

Parameters
error

On input, a pointer to an error object variable. On output, this variable may contain an error object indicating why the bundle's executable could not be loaded. If no error would occur, this parameter is left unmodified. You may specify `nil` for this parameter if you are not interested in the error information.

Return Value
`YES` if the bundle's executable code could be loaded successfully or is already loaded; otherwise, `NO` if the code could not be loaded.

Discussion
This method does not actually load the bundle's executable code. Instead, it performs several checks to see if the code could be loaded and with one exception returns the same errors that would occur during an actual load operation. The one exception is the `NSExecutableLinkError` error, which requires the actual loading of the code to verify link errors.

For a list of possible load errors, see the discussion for the `loadAndReturnError:` (page 93) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- loadAndReturnError:` (page 93)

Declared In
`NSBundle.h`

principalClass
Returns the receiver's principal class.

```
- (Class)principalClass
```

Return Value
The receiver's principal class—after ensuring that the code containing the definition of that class is dynamically loaded. If the receiver encounters errors in loading or if it can't find the executable code file in the bundle directory, returns `NIL`.

Discussion
The principal class typically controls all the other classes in the bundle; it should mediate between those classes and classes external to the bundle. Classes (and categories) are loaded from just one file within the bundle directory. `NSBundle` obtains the name of the code file to load from the dictionary returned from `infoDictionary` (page 91), using "`NSExecutable`" as the key. The bundle determines its principal class in one of two ways:

- It first looks in its own information dictionary, which extracts the information encoded in the bundle's property list (`Info.plist`). `NSBundle` obtains the principal class from the dictionary using the key `NSPrincipalClass`. For nonloadable bundles (applications and frameworks), if the principal class is not specified in the property list, the method returns `NIL`.
- If the principal class is not specified in the information dictionary, `NSBundle` identifies the first class loaded as the principal class. When several classes are linked into a dynamically loadable file, the default principal class is the first one listed on the `ld` command line. In the following example, Reporter would be the principal class:

```
ld -o myBundle -r Reporter.o NotePad.o QueryList.o
```

The order of classes in Xcode's project browser is the order in which they will be linked. To designate the principal class, control-drag the file containing its implementation to the top of the list.

As a side effect of code loading, the receiver posts `NSBundleDidLoadNotification` (page 106) after all classes and categories have been loaded; see "Notifications" (page 106) for details.

The following method obtains a bundle by specifying its path (bundleWithPath: (page 84)), then loads the bundle with principalClass (page 102) and uses the returned class object to allocate and initialize an instance of that class:

```
- (void)loadBundle:(id)sender
{
    Class exampleClass;
    id newInstance;
    NSString *path = @"/tmp/Projects/BundleExample/BundleExample.bundle";
    NSBundle *bundleToLoad = [NSBundle bundleWithPath:path];
    if (exampleClass = [bundleToLoad principalClass]) {
        newInstance = [[exampleClass alloc] init];
        [newInstance doSomething];
    }
}
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- classNamed: (page 89)
- infoDictionary (page 91)
- load (page 93)

Declared In
NSBundle.h

privateFrameworksPath

Returns the full pathname of the receiver's subdirectory containing private frameworks.

```
- (NSString *)privateFrameworksPath
```

Return Value
The full pathname of the receiver's subdirectory containing private frameworks.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

resourcePath

Returns the full pathname of the receiving bundle's subdirectory containing resources.

```
- (NSString *)resourcePath
```

Return Value
The full pathname of the receiving bundle's subdirectory containing resources.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `bundlePath` (page 89)

Declared In
`NSBundle.h`

sharedFrameworksPath

Returns the full pathname of the receiver's subdirectory containing shared frameworks.

```
- (NSString *)sharedFrameworksPath
```

Return Value
The full pathname of the receiver's subdirectory containing shared frameworks.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

sharedSupportPath

Returns the full pathname of the receiver's subdirectory containing shared support files.

```
- (NSString *)sharedSupportPath
```

Return Value
The full pathname of the receiver's subdirectory containing shared support files.

Discussion
This method returns the appropriate path for modern application and framework bundles. This method may not return a path for non-standard bundle formats or for some older bundle formats.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

unload

Unloads the code associated with the receiver.

```
- (BOOL)unload
```

Return Value
YES if the bundle was successfully unloaded or was not already loaded; otherwise, NO if the bundle could not be unloaded.

Discussion
This method attempts to unload a bundle's executable code using the underlying dynamic loader (typically dyld). You may use this method to unload plug-in and framework bundles when you no longer need the code they contain. You should use this method to unload bundles that were loaded using the methods of the NSBundle class only. Do not use this method to unload bundles that were originally loaded using the bundle-manipulation functions in Core Foundation.

It is the responsibility of the caller to ensure that no in-memory objects or data structures refer to the code being unloaded. For example, if you have an object whose class is defined in a bundle, you must release that object prior to unloading the bundle. Similarly, your code should not attempt to access any symbols defined in an unloaded bundle.

Special Considerations
Prior to Mac OS X version 10.5, code could not be unloaded once loaded, and this method would always return NO. In Mac OS X version 10.5 and later, you can unload a bundle's executable code using this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- loadAndReturnError: (page 93)
- load (page 93)

Declared In
NSBundle.h

Constants

Mach-O Architecture
These constants describe the CPU types that a bundle's executable code may support.

```
enum {
    NSBundleExecutableArchitectureI386      = 0x00000007,
    NSBundleExecutableArchitecturePPC       = 0x00000012,
    NSBundleExecutableArchitectureX86_64    = 0x01000007,
    NSBundleExecutableArchitecturePPC64     = 0x01000012
};
```

Constants
NSBundleExecutableArchitectureI386

Specifies the 32-bit Intel architecture.

Available in iPhone OS 2.0 and later.

Declared in NSBundle.h

`NSBundleExecutableArchitecturePPC`
Specifies the 32-bit PowerPC architecture.

Available in iPhone OS 2.0 and later.

Declared in `NSBundle.h`

`NSBundleExecutableArchitectureX86_64`
Specifies the 64-bit Intel architecture.

Available in iPhone OS 2.0 and later.

Declared in `NSBundle.h`

`NSBundleExecutableArchitecturePPC64`
Specifies the 64-bit PowerPC architecture.

Available in iPhone OS 2.0 and later.

Declared in `NSBundle.h`

Declared In
`NSBundle.h`

Notifications

NSBundleDidLoadNotification

`NSBundle` posts `NSBundleDidLoadNotification` to notify observers which classes and categories have been dynamically loaded. When a request is made to an `NSBundle` object for a class (`classNamed:` (page 89) or `principalClass` (page 102)), the bundle dynamically loads the executable code file that contains the class implementation and all other class definitions contained in the file. After the module is loaded, the bundle posts the `NSBundleDidLoadNotification`.

The notification object is the `NSBundle` instance that dynamically loads classes. The *`userInfo`* dictionary contains the following information:

Key	Value
`@"NSLoadedClasses"`	An `NSArray` object containing the names (as `NSString` objects) of each class that was loaded

In a typical use of this notification, an object might want to enumerate the *`userInfo`* array to check if each loaded class conformed to a certain protocol (say, an protocol for a plug-and-play tool set); if a class does conform, the object would create an instance of that class and add the instance to another `NSArray` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

NSCachedURLResponse Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLCache.h
Companion guide:	URL Loading System

Overview

Important: This is a preliminary document for an API or technology in development. Although this document has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSCachedURLResponse` object encapsulates an `NSURLResponse` object, an `NSData` object containing the content corresponding to the response, and an `NSDictionary` containing application specific information.

The `NSURLCache` system stores and retrieves instances of `NSCachedURLResponse`.

Tasks

Creating a Cached URL Response

- `initWithResponse:data:` (page 108)
  Initializes an `NSCachedURLResponse` object.
- `initWithResponse:data:userInfo:storagePolicy:` (page 109)
  Initializes an NSCachedURLResponse object.

Getting Cached URL Response Properties

- `data` (page 108)
  Returns the receiver's cached data.
- `response` (page 110)
  Returns the `NSURLResponse` object associated with the receiver.
- `storagePolicy` (page 110)
  Returns the receiver's cache storage policy.
- `userInfo` (page 110)
  Returns the receiver's user info dictionary.

Instance Methods

data

Returns the receiver's cached data.

```
- (NSData *)data
```

Return Value
The receiver's cached data.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCache.h`

initWithResponse:data:

Initializes an `NSCachedURLResponse` object.

```
- (id)initWithResponse:(NSURLResponse *)response data:(NSData *)data
```

Parameters

response
The response to cache.

data
The data to cache.

Return Value

The `NSCachedURLResponse` object, initialized using the given data.

Discussion

The cache storage policy is set to the default, `NSURLCacheStorageAllowed`, and the user info dictionary is set to `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithResponse:data:userInfo:storagePolicy:` (page 109)

Declared In

`NSURLCache.h`

initWithResponse:data:userInfo:storagePolicy:

Initializes an NSCachedURLResponse object.

```
- (id)initWithResponse:(NSURLResponse *)response data:(NSData *)data
    userInfo:(NSDictionary *)userInfo
    storagePolicy:(NSURLCacheStoragePolicy)storagePolicy
```

Parameters

response
The response to cache.

data
The data to cache.

userInfo
An optional dictionary of user information. May be `nil`.

storagePolicy
The storage policy for the cached response.

Return Value

The `NSCachedURLResponse` object, initialized using the given data.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithResponse:data:` (page 108)

Declared In

`NSURLCache.h`

response

Returns the `NSURLResponse` object associated with the receiver.

```
- (NSURLResponse *)response
```

Return Value
The `NSURLResponse` object associated with the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCache.h`

storagePolicy

Returns the receiver's cache storage policy.

```
- (NSURLCacheStoragePolicy)storagePolicy
```

Return Value
The receiver's cache storage policy.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCache.h`

userInfo

Returns the receiver's user info dictionary.

```
- (NSDictionary *)userInfo
```

Return Value
An `NSDictionary` object containing the receiver's user info, or `nil` if there is no such object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCache.h`

Constants

NSURLCacheStoragePolicy

These constants specify the caching strategy used by an `NSCachedURLResponse` object.

```
typedef enum
{
    NSURLCacheStorageAllowed,
    NSURLCacheStorageAllowedInMemoryOnly,
    NSURLCacheStorageNotAllowed,
} NSURLCacheStoragePolicy;
```

Constants

`NSURLCacheStorageAllowed`

Specifies that storage in `NSURLCache` is allowed without restriction.

Important: iPhone OS ignores this cache policy, and instead treats it as `NSURLCacheStorageAllowedInMemoryOnly`.

Available in iPhone OS 2.0 and later.

Declared in `NSURLCache.h`

`NSURLCacheStorageAllowedInMemoryOnly`

Specifies that storage in `NSURLCache` is allowed; however storage should be restricted to memory only.

Available in iPhone OS 2.0 and later.

Declared in `NSURLCache.h`

`NSURLCacheStorageNotAllowed`

Specifies that storage in `NSURLCache` is not allowed in any fashion, either in memory or on disk.

Available in iPhone OS 2.0 and later.

Declared in `NSURLCache.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLCache.h`

NSCalendar Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSCalendar.h
Companion guides:	Date and Time Programming Guide for Cocoa Data Formatting Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

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.

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 represented by this API may have to map their basic unit concepts into year/month/week/day/…

nomenclature. For example, a calendar composed of 4 quarters in a year instead of 12 months uses the month unit to represent quarters. The particular values of the unit are defined by each calendar, and are not necessarily consistent with values for that unit in another calendar.

To do calendar arithmetic, you use `NSDate` objects in conjunction with a calendar. For example, to convert between a decomposed date in one calendar and another calendar, you must first convert the decomposed elements into a date using the first calendar, then decompose it using the second. `NSDate` provides the absolute scale and epoch (reference point) for dates and times, which can then be rendered into a particular calendar, for calendrical computations or user display.

Two `NSCalendar` methods that return a date object, `dateFromComponents:` (page 120), `dateByAddingComponents:toDate:options:` (page 119), take as a parameter an `NSDateComponents` object that describes the calendrical components required for the computation. You can provide as many components as you need (or choose to). When there is incomplete information to compute an absolute time, default values similar to `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). Related methods (`components:fromDate:` (page 117) and `components:fromDate:toDate:options:` (page 118)) take a bit mask parameter that specifies which components to calculate when returning an `NSDateComponents` object. The bit mask is composed of `NSCalendarUnit` constants (see "Constants" (page 127)).

`NSCalendar` is "toll-free bridged" with its Core Foundation counterpart, CFCalendar. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSCalendar *` parameter, you can pass in a `CFCalendarRef`, and in a function where you see a `CFCalendarRef` parameter, you can pass in an `NSCalendar` instance. See Interchangeable Data Types for more information on toll-free bridging.

Tasks

System Locale Information

`+ currentCalendar` (page 116)
Returns the logical calendar for the current user.

`+ autoupdatingCurrentCalendar` (page 116)
Returns the current logical calendar for the current user.

Initializing a Calendar

`- initWithCalendarIdentifier:` (page 121)
Initializes a newly-allocated `NSCalendar` object for the calendar specified by a given identifier.

`- setFirstWeekday:` (page 125)
Sets the index of the first weekday for the receiver.

`- setLocale:` (page 125)
Sets the locale for the receiver.

`- setMinimumDaysInFirstWeek:` (page 126)
Sets the minimum number of days in the first week of the receiver.

- `setTimeZone:` (page 126)
  Sets the time zone for the receiver.

Getting Information About a Calendar

- `calendarIdentifier` (page 117)
  Returns the identifier for the receiver.
- `firstWeekday` (page 121)
  Returns the index of the first weekday of the receiver.
- `locale` (page 121)
  Returns the locale for the receiver.
- `maximumRangeOfUnit:` (page 122)
  The maximum range limits of the values that a given unit can take on in the receive
- `minimumDaysInFirstWeek` (page 122)
  Returns the minimum number of days in the first week of the receiver.
- `minimumRangeOfUnit:` (page 123)
  Returns the minimum range limits of the values that a given unit can take on in the receiver.
- `ordinalityOfUnit:inUnit:forDate:` (page 123)
  Returns, for a given absolute time, the ordinal number of a smaller calendar unit (such as a day) within a specified larger calendar unit (such as a week).
- `rangeOfUnit:inUnit:forDate:` (page 124)
  Returns the range of absolute time values that a smaller calendar unit (such as a day) can take on in a larger calendar unit (such as a month) that includes a specified absolute time.
- `rangeOfUnit:startDate:interval:forDate:` (page 124)
  Returns by reference the starting time and duration of a given calendar unit that contains a given date.
- `timeZone` (page 127)
  Returns the time zone for the receiver.

Calendrical Calculations

- `components:fromDate:` (page 117)
  Returns a `NSDateComponents` object containing a given date decomposed into specified components.
- `components:fromDate:toDate:options:` (page 118)
  Returns, as an `NSDateComponents` object using specified components, the difference between two supplied dates.
- `dateByAddingComponents:toDate:options:` (page 119)
  Returns a new `NSDate` object representing the absolute time calculated by adding given components to a given date.
- `dateFromComponents:` (page 120)
  Returns a new `NSDate` object representing the absolute time calculated from given components.

Class Methods

autoupdatingCurrentCalendar

Returns the current logical calendar for the current user.

```
+ (id)autoupdatingCurrentCalendar
```

Return Value

The current logical calendar for the current user.

Discussion

Settings you get from this calendar do change as the user's settings change (contrast with `currentCalendar` (page 116)).

Note that if you cache values based on the calendar or related information those caches will of course not be automatically updated by the updating of the calendar object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ currentCalendar` (page 116)
`- initWithCalendarIdentifier:` (page 121)
`- calendarIdentifier` (page 117)

Declared In

`NSCalendar.h`

currentCalendar

Returns the logical calendar for the current user.

```
+ (id)currentCalendar
```

Return Value

The logical calendar for the current user.

Discussion

The returned calendar is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences. Settings you get from this calendar do not change as System Preferences are changed, so that your operations are consistent (contrast with `autoupdatingCurrentCalendar` (page 116)).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ autoupdatingCurrentCalendar` (page 116)
`- initWithCalendarIdentifier:` (page 121)
`- calendarIdentifier` (page 117)

Declared In
NSCalendar.h

Instance Methods

calendarIdentifier

Returns the identifier for the receiver.

```
- (NSString *)calendarIdentifier
```

Return Value
The identifier for the receiver. For valid identifiers, see NSLocale.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ autoupdatingCurrentCalendar (page 116)
- initWithCalendarIdentifier: (page 121)

Declared In
NSCalendar.h

components:fromDate:

Returns a NSDateComponents object containing a given date decomposed into specified components.

```
- (NSDateComponents *)components:(NSUInteger)unitFlags fromDate:(NSDate *)date
```

Parameters
unitFlags
The components into which to decompose *date*—a bitwise OR of NSCalendarUnit constants.

date
The date for which to perform the calculation.

Return Value
An NSDateComponents object containing *date* decomposed into the components specified by *unitFlags*. Returns nil if *date* falls outside of the defined range of the receiver or if the computation cannot be performed

Discussion
The Weekday ordinality, when requested, refers to the next larger (than Week) of the requested units. Some computations can take a relatively long time.

The following example shows how to use this method to determine the current year, month, and day, using an existing calendar (gregorian):

```
unsigned unitFlags = NSYearCalendarUnit | NSMonthCalendarUnit |
NSDayCalendarUnit;
NSDate *date = [NSDate date];
```

```
NSDateComponents *comps = [gregorian components:unitFlags fromDate:date];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `dateFromComponents:` (page 120)
- `components:fromDate:toDate:options:` (page 118)
- `dateByAddingComponents:toDate:options:` (page 119)

Declared In
`NSCalendar.h`

components:fromDate:toDate:options:

Returns, as an `NSDateComponents` object using specified components, the difference between two supplied dates.

```
- (NSDateComponents *)components:(NSUInteger)unitFlags fromDate:(NSDate
    *)startingDate toDate:(NSDate *)resultDate options:(NSUInteger)opts
```

Parameters
unitFlags
: Specifies the components for the returned `NSDateComponents` object—a bitwise `OR` of `NSCalendarUnit` constants.

startingDate
: The start date for the calculation.

resultDate
: The end date for the calculation.

opts
: Options for the calculation.

  If you specify a "wrap" option (`NSWrapCalendarComponents`), the specified components are incremented and wrap around to zero/one on overflow, but do not cause higher units to be incremented. When the wrap option is false, overflow in a unit carries into the higher units, as in typical addition.

Return Value
An `NSDateComponents` object whose components are specified by *unitFlags* and calculated from the difference between the *resultDate* and *startDate* using the options specified by *opts*. Returns `nil` if either date falls outside the defined range of the receiver or if the computation cannot be performed.

Discussion
The result is lossy if there is not a small enough unit requested to hold the full precision of the difference. Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally larger components will be computed before smaller components; for example, in the Gregorian calendar a result might be 1 month and 5 days instead of, for example, 0 months and 35 days. The resulting component values may be negative if *resultDate* is before *startDate*.

The following example shows how to get the approximate number of months and days between two dates using an existing calendar (`gregorian`):

```
NSDate *startDate = ...;
NSDate *endDate = ...;
unsigned int unitFlags = NSMonthCalendarUnit | NSDayCalendarUnit;
NSDateComponents *comps = [gregorian components:unitFlags fromDate:startDate
toDate:endDate  options:0];
int months = [comps month];
int days = [comps day];
```

Note that some computations can take a relatively long time.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `dateByAddingComponents:toDate:options:` (page 119)
- `dateFromComponents:` (page 120)

Declared In
`NSCalendar.h`

dateByAddingComponents:toDate:options:

Returns a new `NSDate` object representing the absolute time calculated by adding given components to a given date.

```
- (NSDate *)dateByAddingComponents:(NSDateComponents *)comps toDate:(NSDate *)date
    options:(NSUInteger)opts
```

Parameters
comps

The components to add to *date*.

date

The date to which *comps* are added.

opts

Options for the calculation. See "`NSDateComponents wrapping behavior`" (page 129) for possible values. Pass `0` to specify no options.

If you specify no options (you pass `0`), overflow in a unit carries into the higher units (as in typical addition).

Return Value
A new `NSDate` object representing the absolute time calculated by adding to *date* the calendrical components specified by *comps* using the options specified by *opts*. Returns `nil` if *date* falls outside the defined range of the receiver or if the computation cannot be performed.

Discussion
Some operations can be ambiguous, and the behavior of the computation is calendar-specific, but generally components are added in the order specified.

The following example shows how to add 2 months and 3 days to the current date and time using an existing calendar (`gregorian`):

```
NSDate *currentDate = [NSDate date];
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setMonth:2];
```

```
[comps setDay:3];
NSDate *date = [gregorian dateByAddingComponents:comps toDate:currentDate
options:0];
[comps release];
```

Note that some computations can take a relatively long time.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `dateFromComponents:` (page 120)
- `components:fromDate:toDate:options:` (page 118)

Declared In
`NSCalendar.h`

dateFromComponents:

Returns a new `NSDate` object representing the absolute time calculated from given components.

```
- (NSDate *)dateFromComponents:(NSDateComponents *)comps
```

Parameters
comps

The components from which to calculate the returned date.

Return Value
A new `NSDate` object representing the absolute time calculated from *comps*. Returns `nil` if the receiver cannot convert the components given in *comps* into an absolute time. The method also returns `nil` and for out-of-range values.

Discussion
When there are insufficient components provided to completely specify an absolute time, a calendar uses default values of its choice. When there is inconsistent information, a calendar may ignore some of the components parameters or the method may return `nil`. Unnecessary components are ignored (for example, Day takes precedence over Weekday and Weekday ordinals).

The following example shows how to use this method to create a date object to represent 14:10:00 on 6 January 1965, for a given calendar (`gregorian`).

```
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setYear:1965];
[comps setMonth:1];
[comps setDay:6];
[comps setHour:14];
[comps setMinute:10];
[comps setSecond:0];
NSDate *date = [gregorian dateFromComponents:comps];
[comps release];
```

Note that some computations can take a relatively long time to perform.

Availability
Available in iPhone OS 2.0 and later.

See Also
- components:fromDate: (page 117)
- dateFromComponents: (page 120)

Declared In
NSCalendar.h

firstWeekday

Returns the index of the first weekday of the receiver.

```
- (NSUInteger)firstWeekday
```

Return Value
The index of the first weekday of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setFirstWeekday: (page 125)

Declared In
NSCalendar.h

initWithCalendarIdentifier:

Initializes a newly-allocated NSCalendar object for the calendar specified by a given identifier.

```
- (id)initWithCalendarIdentifier:(NSString *)string
```

Parameters
string
The identifier for the new calendar. For valid identifiers, see NSLocale.

Return Value
The initialized calendar, or nil if the identifier is unknown (if, for example, it is either an unrecognized string or the calendar is not supported by the current version of the operating system).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ autoupdatingCurrentCalendar (page 116)
- calendarIdentifier (page 117)

Declared In
NSCalendar.h

locale

Returns the locale for the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setLocale: (page 125)

Declared In
NSCalendar.h

maximumRangeOfUnit:

The maximum range limits of the values that a given unit can take on in the receive

```
- (NSRange)maximumRangeOfUnit:(NSCalendarUnit)unit
```

Parameters
unit
The unit for which the maximum range is returned.

Return Value
The maximum range limits of the values that the unit specified by *unit* can take on in the receiver.

Discussion
As an example, in the Gregorian calendar the maximum range of values for the Day unit is 1-31.

Availability
Available in iPhone OS 2.0 and later.

See Also
- minimumRangeOfUnit: (page 123)

Declared In
NSCalendar.h

minimumDaysInFirstWeek

Returns the minimum number of days in the first week of the receiver.

```
- (NSUInteger)minimumDaysInFirstWeek
```

Return Value
The minimum number of days in the first week of the receiver

Availability
Available in iPhone OS 2.0 and later.

See Also
- setMinimumDaysInFirstWeek: (page 126)

Declared In
NSCalendar.h

minimumRangeOfUnit:

Returns the minimum range limits of the values that a given unit can take on in the receiver.

```
- (NSRange)minimumRangeOfUnit:(NSCalendarUnit)unit
```

Parameters

unit

The unit for which the maximum range is returned.

Return Value
The minimum range limits of the values that the unit specified by *unit* can take on in the receiver.

Discussion
As an example, in the Gregorian calendar the minimum range of values for the Day unit is 1-28.

Availability
Available in iPhone OS 2.0 and later.

See Also
- maximumRangeOfUnit: (page 122)

Declared In
NSCalendar.h

ordinalityOfUnit:inUnit:forDate:

Returns, for a given absolute time, the ordinal number of a smaller calendar unit (such as a day) within a specified larger calendar unit (such as a week).

```
- (NSUInteger)ordinalityOfUnit:(NSCalendarUnit)smaller inUnit:(NSCalendarUnit)larger
    forDate:(NSDate *)date
```

Parameters

smaller

The smaller calendar unit

larger

The larger calendar unit

date

The absolute time for which the calculation is performed

Return Value
The ordinal number of *smaller* within *larger* at the time specified by *date*. Returns NSNotFound if *larger* is not logically bigger than *smaller* in the calendar, or the given combination of units does not make sense (or is a computation which is undefined).

Discussion
The ordinality is in most cases not the same as the decomposed value of the unit. Normal return values are 1 and greater. For example, the time 00:45 is in the first hour of the day, and for units Hour and Day respectively, the result would be 1.

Note that some computations can take a relatively long time.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `rangeOfUnit:inUnit:forDate:` (page 124)
- `rangeOfUnit:startDate:interval:forDate:` (page 124)

Declared In
`NSCalendar.h`

rangeOfUnit:inUnit:forDate:

Returns the range of absolute time values that a smaller calendar unit (such as a day) can take on in a larger calendar unit (such as a month) that includes a specified absolute time.

```
- (NSRange)rangeOfUnit:(NSCalendarUnit)smaller inUnit:(NSCalendarUnit)larger
    forDate:(NSDate *)date
```

Parameters
smaller
The smaller calendar unit.

larger
The larger calendar unit.

date
The absolute time for which the calculation is performed.

Return Value
The range of absolute time values *smaller* can take on in *larger* at the time specified by *date*. Returns {NSNotFound, NSNotFound} if *larger* is not logically bigger than *smaller* in the calendar, or the given combination of units does not make sense (or is a computation which is undefined).

Discussion
You can use this method to calculate, for example, the range the Day unit can take on in the Month in which *date* lies.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `rangeOfUnit:startDate:interval:forDate:` (page 124)
- `ordinalityOfUnit:inUnit:forDate:` (page 123)

Declared In
`NSCalendar.h`

rangeOfUnit:startDate:interval:forDate:

Returns by reference the starting time and duration of a given calendar unit that contains a given date.

```
- (BOOL)rangeOfUnit:(NSCalendarUnit)unit startDate:(NSDate **)datep
    interval:(NSTimeInterval *)tip forDate:(NSDate *)date
```

Parameters

unit

A calendar unit (see "Calendar Units" (page 127) for possible values).

datep

Upon return, contains the starting time of the calendar unit *unit* that contains the date *date*

tip

Upon return, contains the duration of the calendar unit *unit* that contains the date *date*

date

A date.

Return Value

`YES` if the starting time and duration of a unit could be calculated, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `rangeOfUnit:inUnit:forDate:` (page 124)
- `ordinalityOfUnit:inUnit:forDate:` (page 123)

Declared In

`NSCalendar.h`

setFirstWeekday:

Sets the index of the first weekday for the receiver.

```
- (void)setFirstWeekday:(NSUInteger)weekday
```

Parameters

weekday

The first weekday for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `firstWeekday` (page 121)

Declared In

`NSCalendar.h`

setLocale:

Sets the locale for the receiver.

```
- (void)setLocale:(NSLocale *)locale
```

Parameters

locale

The locale for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- locale (page 121)

Declared In

NSCalendar.h

setMinimumDaysInFirstWeek:

Sets the minimum number of days in the first week of the receiver.

```
- (void)setMinimumDaysInFirstWeek:(NSUInteger)mdw
```

Parameters

mdw

The minimum number of days in the first week of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- minimumDaysInFirstWeek (page 122)

Declared In

NSCalendar.h

setTimeZone:

Sets the time zone for the receiver.

```
- (void)setTimeZone:(NSTimeZone *)tz
```

Parameters

tz

The time zone for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- timeZone (page 127)

Declared In

NSCalendar.h

timeZone

Returns the time zone for the receiver.

```
- (NSTimeZone *)timeZone
```

Return Value
The time zone for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setTimeZone: (page 126)

Declared In
NSCalendar.h

Constants

NSCalendarUnit

Defines a type used to specify calendrical units such as day and month.

```
typedef NSUInteger NSCalendarUnit;
```

Discussion
See "Calendar Units" (page 127) for possible values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCalendar.h

Calendar Units

Specify calendrical units such as day and month.

```
enum {
    NSEraCalendarUnit = kCFCalendarUnitEra,
    NSYearCalendarUnit = kCFCalendarUnitYear,
    NSMonthCalendarUnit = kCFCalendarUnitMonth,
    NSDayCalendarUnit = kCFCalendarUnitDay,
    NSHourCalendarUnit = kCFCalendarUnitHour,
    NSMinuteCalendarUnit = kCFCalendarUnitMinute,
    NSSecondCalendarUnit = kCFCalendarUnitSecond,
    NSWeekCalendarUnit = kCFCalendarUnitWeek,
    NSWeekdayCalendarUnit = kCFCalendarUnitWeekday,
    NSWeekdayOrdinalCalendarUnit = kCFCalendarUnitWeekdayOrdinal
};
```

Constants

`NSEraCalendarUnit`

Specifies the era unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitEra`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSYearCalendarUnit`

Specifies the year unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitYear`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSMonthCalendarUnit`

Specifies the month unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitMonth`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSDayCalendarUnit`

Specifies the day unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitDay`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSHourCalendarUnit`

Specifies the hour unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitHour`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSMinuteCalendarUnit`

Specifies the minute unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitMinute`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSSecondCalendarUnit`
Specifies the second unit.

The corresponding value is a `double`. Equal to `kCFCalendarUnitSecond`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSWeekCalendarUnit`
Specifies the week unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeek`.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSWeekdayCalendarUnit`
Specifies the weekday unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeekday`. The weekday units are the numbers 1 through N (where for the Gregorian calendar N=7 and 1 is Sunday).

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

`NSWeekdayOrdinalCalendarUnit`
Specifies the ordinal weekday unit.

The corresponding value is an `int`. Equal to `kCFCalendarUnitWeekdayOrdinal`. The weekday ordinal unit describes ordinal position within the month unit of the corresponding weekday unit. For example, in the Gregorian calendar a weekday ordinal unit of 2 for a weekday unit 3 indicates "the second Tuesday in the month".

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

Discussion
Calendar units may be used as a bit mask to specify a combination of units. Values in this `enum` are equal to the corresponding constants in the `CFCalendarUnit enum`.

Declared In
`NSCalendar.h`

NSDateComponents wrapping behavior

The wrapping option specifies wrapping behavior for calculations involving `NSDateComponents` objects.

```
enum
{
    NSWrapCalendarComponents = kCFCalendarComponentsWrap,
};
```

Constants
`NSWrapCalendarComponents`
Specifies that the components specified for an `NSDateComponents` object should be incremented and wrap around to zero/one on overflow, but should not cause higher units to be incremented.

Available in iPhone OS 2.0 and later.

Declared in `NSCalendar.h`

Declared In
NSCalendar.h

NSCharacterSet Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSCharacterSet.h
Companion guide:	String Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSCharacterSet` object represents a set of Unicode-compliant characters. `NSString` and `NSScanner` objects use `NSCharacterSet` objects to group characters together for searching operations, so that they can find any of a particular set of characters during a search. The cluster's two public classes, `NSCharacterSet` and `NSMutableCharacterSet`, declare the programmatic interface for static and dynamic character sets, respectively.

The objects you create using these classes are referred to as character set objects (and when no confusion will result, merely as character sets). Because of the nature of class clusters, character set objects aren't actual instances of the `NSCharacterSet` or `NSMutableCharacterSet` classes but of one of their private subclasses. Although a character set object's class is private, its interface is public, as declared by these abstract superclasses, `NSCharacterSet` and `NSMutableCharacterSet`. The character set classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a character set of one type to the other.

The `NSCharacterSet` class declares the programmatic interface for an object that manages a set of Unicode characters (see the `NSString` class cluster specification for information on Unicode). `NSCharacterSet`'s principal primitive method, `characterIsMember:` (page 143), provides the basis for all other instance methods in its interface. A subclass of `NSCharacterSet` needs only to implement this method, plus `mutableCopyWithZone:` (page 1300), for proper behavior. For optimal performance, a subclass should also override `bitmapRepresentation` (page 143), which otherwise works by invoking `characterIsMember:` (page 143) for every possible Unicode value.

`NSCharacterSet` is "toll-free bridged" with its Cocoa Foundation counterpart, CFCharacterSet. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSCharacterSet *` parameter, you can pass a `CFCharacterSetRef`, and in a function where you see a `CFCharacterSetRef` parameter, you can pass an `NSCharacterSet` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

The mutable subclass of `NSCharacterSet` is `NSMutableCharacterSet`.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

NSMutableCopying
- `mutableCopyWithZone:` (page 1300)

Tasks

Creating a Standard Character Set

`+ alphanumericCharacterSet` (page 134)
Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

`+ capitalizedLetterCharacterSet` (page 135)
Returns a character set containing the characters in the category of Titlecase Letters.

+ controlCharacterSet (page 137)
  Returns a character set containing the characters in the categories of Control or Format Characters.

+ decimalDigitCharacterSet (page 138)
  Returns a character set containing the characters in the category of Decimal Numbers.

+ decomposableCharacterSet (page 138)
  Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

+ illegalCharacterSet (page 138)
  Returns a character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

+ letterCharacterSet (page 139)
  Returns a character set containing the characters in the categories Letters and Marks.

+ lowercaseLetterCharacterSet (page 139)
  Returns a character set containing the characters in the category of Lowercase Letters.

+ newlineCharacterSet (page 140)
  Returns a character set containing the newline characters.

+ nonBaseCharacterSet (page 140)
  Returns a character set containing the characters in the category of Marks.

+ punctuationCharacterSet (page 140)
  Returns a character set containing the characters in the category of Punctuation.

+ symbolCharacterSet (page 141)
  Returns a character set containing the characters in the category of Symbols.

+ uppercaseLetterCharacterSet (page 141)
  Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

+ whitespaceAndNewlineCharacterSet (page 142)
  Returns a character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline and nextline characters (U+000A–U+000D, U+0085).

+ whitespaceCharacterSet (page 142)
  Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

Creating a Custom Character Set

+ characterSetWithCharactersInString: (page 136)
  Returns a character set containing the characters in a given string.

+ characterSetWithRange: (page 137)
  Returns a character set containing characters with Unicode values in a given range.

- invertedSet (page 144)
  Returns a character set containing only characters that don't exist in the receiver.

Creating and Managing Character Sets as Bitmap Representations

`+ characterSetWithBitmapRepresentation:` (page 135)
Returns a character set containing characters determined by a given bitmap representation.

`+ characterSetWithContentsOfFile:` (page 136)
Returns a character set read from the bitmap representation stored in the file a given path.

`- bitmapRepresentation` (page 143)
Returns an `NSData` object encoding the receiver in binary format.

Testing Set Membership

`- characterIsMember:` (page 143)
Returns a Boolean value that indicates whether a given character is in the receiver.

`- hasMemberInPlane:` (page 144)
Returns a Boolean value that indicates whether the receiver has at least one member in a given character plane.

`- isSupersetOfSet:` (page 144)
Returns a Boolean value that indicates whether the receiver is a superset of another given character set.

`- longCharacterIsMember:` (page 145)
Returns a Boolean value that indicates whether a given long character is a member of the receiver.

Class Methods

alphanumericCharacterSet

Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

```
+ (id)alphanumericCharacterSet
```

Return Value

A character set containing the characters in the categories Letters, Marks, and Numbers.

Discussion

Informally, this set is the set of all characters used as basic units of alphabets, syllabaries, ideographs, and digits.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ letterCharacterSet` (page 139)
`+ decimalDigitCharacterSet` (page 138)

Declared In

`NSCharacterSet.h`

capitalizedLetterCharacterSet

Returns a character set containing the characters in the category of Titlecase Letters.

```
+ (id)capitalizedLetterCharacterSet
```

Return Value
A character set containing the characters in the category of Titlecase Letters.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ letterCharacterSet (page 139)
+ uppercaseLetterCharacterSet (page 141)

Declared In
NSCharacterSet.h

characterSetWithBitmapRepresentation:

Returns a character set containing characters determined by a given bitmap representation.

```
+ (id)characterSetWithBitmapRepresentation:(NSData *)data
```

Parameters
data

A bitmap representation of a character set.

Return Value
A character set containing characters determined by *data*.

Discussion
This method is useful for creating a character set object with data from a file or other external data source.

A raw bitmap representation of a character set is a byte array of 2^16 bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value *n*. To add a character with decimal Unicode value *n* to a raw bitmap representation, use a statement such as the following:

```
unsigned char bitmapRep[8192];
bitmapRep[n >> 3] |= (((unsigned int)1) << (n & 7));
```

To remove that character:

```
bitmapRep[n >> 3] &= ~(((unsigned int)1) << (n & 7));
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- bitmapRepresentation (page 143)
+ characterSetWithContentsOfFile: (page 136)

Declared In
NSCharacterSet.h

characterSetWithCharactersInString:

Returns a character set containing the characters in a given string.

```
+ (id)characterSetWithCharactersInString:(NSString *)aString
```

Parameters

aString

A string containing characters for the new character set.

Return Value
A character set containing the characters in *aString*. Returns an empty character set if *aString* is empty.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCharacterSet.h

characterSetWithContentsOfFile:

Returns a character set read from the bitmap representation stored in the file a given path.

```
+ (id)characterSetWithContentsOfFile:(NSString *)path
```

Parameters

path

A path to a file containing a bitmap representation of a character set. The path name must end with the extension `.bitmap`.

Return Value
A character set read from the bitmap representation stored in the file at *path*.

Discussion
To read a bitmap representation from any file, use the `NSData` method`dataWithContentsOfFile:options:error:` (page 195) and pass the result to `characterSetWithBitmapRepresentation:` (page 135).

This method doesn't use filenames to check for the uniqueness of the character sets it creates. To prevent duplication of character sets in memory, cache them and make them available through an API that checks whether the requested set has already been loaded.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCharacterSet.h

characterSetWithRange:

Returns a character set containing characters with Unicode values in a given range.

```
+ (id)characterSetWithRange:(NSRange)aRange
```

Parameters

aRange

A range of Unicode values.

`aRange.location` is the value of the first character to remove; `aRange.location + aRange.length- 1` is the value of the last.

Return Value

A character set containing characters whose Unicode values are given by *aRange*. If `aRange.length` is `0`, returns an empty character set.

Discussion

This code excerpt creates a character set object containing the lowercase English alphabetic characters:

```
NSRange lcEnglishRange;
NSCharacterSet *lcEnglishLetters;

lcEnglishRange.location = (unsigned int)'a';
lcEnglishRange.length = 26;
lcEnglishLetters = [NSCharacterSet characterSetWithRange:lcEnglishRange];
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSCharacterSet.h`

controlCharacterSet

Returns a character set containing the characters in the categories of Control or Format Characters.

```
+ (id)controlCharacterSet
```

Return Value

A character set containing the characters in the categories of Control or Format Characters.

Discussion

These characters are specifically the Unicode values `U+0000` to `U+001F` and `U+007F` to `U+009F`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ illegalCharacterSet` (page 138)

Declared In

`NSCharacterSet.h`

decimalDigitCharacterSet

Returns a character set containing the characters in the category of Decimal Numbers.

```
+ (id)decimalDigitCharacterSet
```

Return Value
A character set containing the characters in the category of Decimal Numbers.

Discussion
Informally, this set is the set of all characters used to represent the decimal values `0` through `9`. These characters include, for example, the decimal digits of the Indic scripts and Arabic.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ alphanumericCharacterSet` (page 134)

Declared In
`NSCharacterSet.h`

decomposableCharacterSet

Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

```
+ (id)decomposableCharacterSet
```

Return Value
A character set containing all individual Unicode characters that can also be represented as composed character sequences (such as for letters with accents), by the definition of “standard decomposition” in version 3.2 of the Unicode character encoding standard.

Discussion
These characters include compatibility characters as well as pre-composed characters.

Note: This character set doesn’t currently include the Hangul characters defined in version 2.0 of the Unicode standard.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ nonBaseCharacterSet` (page 140)

Declared In
`NSCharacterSet.h`

illegalCharacterSet

Returns a character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

```
+ (id)illegalCharacterSet
```

Return Value
A character set containing values in the category of Non-Characters or that have not yet been defined in version 3.2 of the Unicode standard.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ controlCharacterSet (page 137)

Declared In
NSCharacterSet.h

letterCharacterSet

Returns a character set containing the characters in the categories Letters and Marks.

```
+ (id)letterCharacterSet
```

Return Value
A character set containing the characters in the categories Letters and Marks.

Discussion
Informally, this set is the set of all characters used as letters of alphabets and ideographs.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ alphanumericCharacterSet (page 134)
+ lowercaseLetterCharacterSet (page 139)
+ uppercaseLetterCharacterSet (page 141)

Declared In
NSCharacterSet.h

lowercaseLetterCharacterSet

Returns a character set containing the characters in the category of Lowercase Letters.

```
+ (id)lowercaseLetterCharacterSet
```

Return Value
A character set containing the characters in the category of Lowercase Letters.

Discussion
Informally, this set is the set of all characters used as lowercase letters in alphabets that make case distinctions.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ uppercaseLetterCharacterSet (page 141)
+ letterCharacterSet (page 139)

Declared In
NSCharacterSet.h

newlineCharacterSet

Returns a character set containing the newline characters.

```
+ (id)newlineCharacterSet
```

Return Value
A character set containing the newline characters (U+000A–U+000D, U+0085).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ whitespaceAndNewlineCharacterSet (page 142)
+ whitespaceCharacterSet (page 142)

Declared In
NSCharacterSet.h

nonBaseCharacterSet

Returns a character set containing the characters in the category of Marks.

```
+ (id)nonBaseCharacterSet
```

Return Value
A character set containing the characters in the category of Marks.

Discussion
This set is also defined as all legal Unicode characters with a non-spacing priority greater than 0. Informally, this set is the set of all characters used as modifiers of base characters.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ decomposableCharacterSet (page 138)

Declared In
NSCharacterSet.h

punctuationCharacterSet

Returns a character set containing the characters in the category of Punctuation.

```
+ (id)punctuationCharacterSet
```

Return Value
A character set containing the characters in the category of Punctuation.

Discussion
Informally, this set is the set of all non-whitespace characters used to separate linguistic units in scripts, such as periods, dashes, parentheses, and so on.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCharacterSet.h

symbolCharacterSet

Returns a character set containing the characters in the category of Symbols.

```
+ (id)symbolCharacterSet
```

Return Value
A character set containing the characters in the category of Symbols.

Discussion
These characters include, for example, the dollar sign ($) and the plus (+) sign.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCharacterSet.h

uppercaseLetterCharacterSet

Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

```
+ (id)uppercaseLetterCharacterSet
```

Return Value
A character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

Discussion
Informally, this set is the set of all characters used as uppercase letters in alphabets that make case distinctions.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ capitalizedLetterCharacterSet (page 135)
+ lowercaseLetterCharacterSet (page 139)
+ letterCharacterSet (page 139)

Declared In
NSCharacterSet.h

whitespaceAndNewlineCharacterSet

Returns a character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline and nextline characters (U+000A–U+000D, U+0085).

```
+ (id)whitespaceAndNewlineCharacterSet
```

Return Value
A character set containing only the whitespace characters space (U+0020) and tab (U+0009) and the newline and nextline characters (U+000A–U+000D, U+0085).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ newlineCharacterSet (page 140)
+ whitespaceCharacterSet (page 142)

Declared In
NSCharacterSet.h

whitespaceCharacterSet

Returns a character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

```
+ (id)whitespaceCharacterSet
```

Return Value
A character set containing only the in-line whitespace characters space (U+0020) and tab (U+0009).

Discussion
This set doesn't contain the newline or carriage return characters.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ whitespaceAndNewlineCharacterSet (page 142)
+ newlineCharacterSet (page 140)

Declared In
NSCharacterSet.h

Instance Methods

bitmapRepresentation

Returns an `NSData` object encoding the receiver in binary format.

```
- (NSData *)bitmapRepresentation
```

Return Value

An `NSData` object encoding the receiver in binary format.

Discussion

This format is suitable for saving to a file or otherwise transmitting or archiving.

A raw bitmap representation of a character set is a byte array of 2^16 bits (that is, 8192 bytes). The value of the bit at position *n* represents the presence in the character set of the character with decimal Unicode value *n*. To test for the presence of a character with decimal Unicode value *n* in a raw bitmap representation, use an expression such as the following:

```
unsigned char bitmapRep[8192];
if (bitmapRep[n >> 3] & (((unsigned int)1) << (n  & 7))) {
    /* Character is present. */
}
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ characterSetWithBitmapRepresentation:` (page 135)

Declared In

`NSCharacterSet.h`

characterIsMember:

Returns a Boolean value that indicates whether a given character is in the receiver.

```
- (BOOL)characterIsMember:(unichar)aCharacter
```

Parameters

aCharacter

The character to test for membership of the receiver.

Return Value

`YES` if *aCharacter* is in the receiving character set, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- longCharacterIsMember:` (page 145)

Declared In
NSCharacterSet.h

hasMemberInPlane:

Returns a Boolean value that indicates whether the receiver has at least one member in a given character plane.

```
- (BOOL)hasMemberInPlane:(uint8_t)thePlane
```

Parameters

thePlane
A character plane.

Return Value
YES if the receiver has at least one member in *thePlane*, otherwise NO.

Discussion
This method makes it easier to find the plane containing the members of the current character set. The Basic Multilingual Plane is plane 0.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCharacterSet.h

invertedSet

Returns a character set containing only characters that don't exist in the receiver.

```
- (NSCharacterSet *)invertedSet
```

Return Value
A character set containing only characters that don't exist in the receiver.

Discussion
Inverting an immutable character set is much more efficient than inverting a mutable character set.

Availability
Available in iPhone OS 2.0 and later.

See Also
invert (page 590) (NSMutableCharacterSet)

Declared In
NSCharacterSet.h

isSupersetOfSet:

Returns a Boolean value that indicates whether the receiver is a superset of another given character set.

```
- (BOOL)isSupersetOfSet:(NSCharacterSet *)theOtherSet
```

Parameters

theOtherSet

A character set.

Return Value

YES if the receiver is a superset of *theOtherSet*, otherwise NO.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSCharacterSet.h

longCharacterIsMember:

Returns a Boolean value that indicates whether a given long character is a member of the receiver.

```
- (BOOL)longCharacterIsMember:(UTF32Char)theLongChar
```

Parameters

theLongChar

A UTF32 character.

Return Value

YES if *theLongChar* is in the receiver, otherwise NO.

Discussion

This method supports the specification of 32-bit characters.

Availability

Available in iPhone OS 2.0 and later.

See Also

- characterIsMember: (page 143)

Declared In

NSCharacterSet.h

Constants

NSOpenStepUnicodeReservedBase

Specifies lower bound for a Unicode character range reserved for Apple's corporate use.

```
enum {
   NSOpenStepUnicodeReservedBase = 0xF400
};
```

Constants

`NSOpenStepUnicodeReservedBase`

Specifies lower bound for a Unicode character range reserved for Apple's corporate use (the range is `0xF400-0xF8FF`).

Available in iPhone OS 2.0 and later.

Declared in `NSCharacterSet.h`

Declared In

`NSCharacterSet.h`

NSCoder Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSCoder.h Foundation/NSKeyedArchiver.h Foundation/NSGeometry.h
Companion guide:	Archives and Serializations Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSCoder` abstract class declares the interface used by concrete subclasses to transfer objects and other Objective-C data items between memory and some other format. This capability provides the basis for archiving (where objects and data items are stored on disk) and distribution (where objects and data items are copied between different processes or threads). The concrete subclasses provided by Foundation for these purposes are `NSArchiver`, `NSUnarchiver`, `NSKeyedArchiver`, `NSKeyedUnarchiver`, and `NSPortCoder`. Concrete subclasses of `NSCoder` are referred to in general as coder classes, and instances of these classes as coder objects (or simply coders). A coder object that can only encode values is referred to as an encoder object, and one that can only decode values as a decoder object.

`NSCoder` operates on objects, scalars, C arrays, structures, and strings, and on pointers to these types. It does not handle types whose implementation varies across platforms, such as `union`, `void *`, function pointers, and long chains of pointers. A coder object stores object type information along with the data, so an object decoded from a stream of bytes is normally of the same class as the object that was originally encoded into the stream. An object can change its class when encoded, however; this is described in *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Testing Coder

- `allowsKeyedCoding` (page 151)
  Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.
- `containsValueForKey:` (page 151)
  Returns a Boolean value that indicates whether an encoded value is available for a string.

Encoding Data

- `encodeArrayOfObjCType:count:at:` (page 157)
  Encodes an array of *`count`* items, whose Objective-C type is given by *`itemType`*.
- `encodeBool:forKey:` (page 158)
  Encodes *`boolv`* and associates it with the string *`key`*.
- `encodeBycopyObject:` (page 158)
  Can be overridden by subclasses to encode *`object`* so that a copy, rather than a proxy, is created upon decoding.
- `encodeByrefObject:` (page 159)
  Can be overridden by subclasses to encode *`object`* so that a proxy, rather than a copy, is created upon decoding.
- `encodeBytes:length:` (page 159)
  Encodes a buffer of data whose types are unspecified.
- `encodeBytes:length:forKey:` (page 160)
  Encodes a buffer of data, *`bytesp`*, whose length is specified by *`lenv`*, and associates it with the string *`key`*.
- `encodeConditionalObject:` (page 160)
  Can be overridden by subclasses to conditionally encode *`object`*, preserving common references to that object.
- `encodeConditionalObject:forKey:` (page 161)
  Conditionally encodes a reference to *`objv`* and associates it with the string *`key`* only if *`objv`* has been unconditionally encoded with `encodeObject:forKey:` (page 164).
- `encodeDataObject:` (page 161)
  Encodes a given `NSData` object.
- `encodeDouble:forKey:` (page 161)
  Encodes *`realv`* and associates it with the string *`key`*.

- `encodeFloat:forKey:` (page 162)
  Encodes *realv* and associates it with the string *key*.
- `encodeInt:forKey:` (page 163)
  Encodes *intv* and associates it with the string *key*.
- `encodeInteger:forKey:` (page 163)
  Encodes a given `NSInteger` and associates it with a given key.
- `encodeInt32:forKey:` (page 162)
  Encodes the 32-bit integer *intv* and associates it with the string *key*.
- `encodeInt64:forKey:` (page 162)
  Encodes the 64-bit integer *intv* and associates it with the string *key*.
- `encodeObject:` (page 164)
  Encodes *object*.
- `encodeObject:forKey:` (page 164)
  Encodes the object *objv* and associates it with the string *key*.
- `encodeRootObject:` (page 164)
  Can be overridden by subclasses to encode an interconnected group of Objective-C objects, starting with *rootObject*.
- `encodeValueOfObjCType:at:` (page 165)
  Must be overridden by subclasses to encode a single value residing at *address*, whose Objective-C type is given by *valueType*.
- `encodeValuesOfObjCTypes:` (page 165)
  Encodes a series of values of potentially differing Objective-C types.

Decoding Data

- `decodeArrayOfObjCType:count:at:` (page 151)
  Decodes an array of *count* items, whose Objective-C type is given by *itemType*.
- `decodeBoolForKey:` (page 152)
  Decodes and returns a boolean value that was previously encoded with `encodeBool:forKey:` (page 158) and associated with the string *key*.
- `decodeBytesForKey:returnedLength:` (page 152)
  Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 160) and associated with the string *key*.
- `decodeBytesWithReturnedLength:` (page 153)
  Decodes a buffer of data whose types are unspecified.
- `decodeDataObject` (page 153)
  Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 161). Subclasses must override this method.
- `decodeDoubleForKey:` (page 153)
  Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 162) or `encodeDouble:forKey:` (page 161) and associated with the string *key*.

- `decodeFloatForKey:` (page 154)
  Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 162) or `encodeDouble:forKey:` (page 161) and associated with the string *key*.
- `decodeIntForKey:` (page 155)
  Decodes and returns an `int` value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.
- `decodeIntegerForKey:` (page 155)
  Decodes and returns an `NSInteger` value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.
- `decodeInt32ForKey:` (page 154)
  Decodes and returns a 32-bit integer value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.
- `decodeInt64ForKey:` (page 154)
  Decodes and returns a 64-bit integer value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.
- `decodeObject` (page 155)
  Decodes an Objective-C object that was previously encoded with any of the `encode...Object:` methods.
- `decodeObjectForKey:` (page 156)
  Decodes and returns an autoreleased Objective-C object that was previously encoded with `encodeObject:forKey:` (page 164) or `encodeConditionalObject:forKey:` (page 161) and associated with the string *key*.
- `decodeValueOfObjCType:at:` (page 156)
  Decodes a single value, whose Objective-C type is given by *valueType*.
- `decodeValuesOfObjCTypes:` (page 157)
  Decodes a series of potentially different Objective-C types.

Managing Zones

- `objectZone` (page 166)
  Returns the memory zone used to allocate decoded objects.
- `setObjectZone:` (page 166)
  `NSCoder`'s implementation of this method does nothing.

Getting Version Information

- `systemVersion` (page 167)
  During encoding, this method should return the system version currently in effect.

- versionForClassName: (page 167)
  Returns the version in effect for the class with a given name.

Instance Methods

allowsKeyedCoding

Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.

```
- (BOOL)allowsKeyedCoding
```

Discussion

The default implementation returns NO. Concrete subclasses that support keyed coding, such as NSKeyedArchiver, need to override this method to return YES.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSCoder.h

containsValueForKey:

Returns a Boolean value that indicates whether an encoded value is available for a string.

```
- (BOOL)containsValueForKey:(NSString *)key
```

Discussion

The string is passed as *key*. Subclasses must override this method if they perform keyed coding.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSCoder.h

decodeArrayOfObjCType:count:at:

Decodes an array of *count* items, whose Objective-C type is given by *itemType*.

```
- (void)decodeArrayOfObjCType:(const char *)itemType count:(NSUInteger)count at:(void
    *)address
```

Discussion

The items are decoded into the buffer beginning at *address*, which must be large enough to contain them all. *itemType* must contain exactly one type code. NSCoder's implementation invokes decodeValueOfObjCType:at: (page 156) to decode the entire array of items. If you use this method to decode an array of Objective-C objects, you are responsible for releasing each object.

This method matches an `encodeArrayOfObjCType:count:at:` (page 157) message used during encoding.

For information on creating an Objective-C type code suitable for *itemType*, see the "Type Encodings" section in the "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeValuesOfObjCTypes:` (page 157)

Declared In
`NSCoder.h`

decodeBoolForKey:

Decodes and returns a boolean value that was previously encoded with `encodeBool:forKey:` (page 158) and associated with the string *key*.

```
- (BOOL)decodeBoolForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSCoder.h`

decodeBytesForKey:returnedLength:

Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 160) and associated with the string *key*.

```
- (const uint8_t *)decodeBytesForKey:(NSString *)key returnedLength:(NSUInteger
    *)lengthp
```

Discussion
The buffer's length is returned by reference in *lengthp*. The returned bytes are immutable. Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeBytes:length:forKey:` (page 160)

Declared In
`NSCoder.h`

decodeBytesWithReturnedLength:

Decodes a buffer of data whose types are unspecified.

```
- (void *)decodeBytesWithReturnedLength:(NSUInteger *)numBytes
```

Discussion

`NSCoder`'s implementation invokes `decodeValueOfObjCType:at:` (page 156) to decode the data as a series of bytes, which this method then places into a buffer and returns. The buffer's length is returned by reference in *numBytes*. If you need the bytes beyond the scope of the current autorelease pool, you must copy them.

This method matches an `encodeBytes:length:` (page 159) message used during encoding.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeArrayOfObjCType:count:at:` (page 157)

Declared In

`NSCoder.h`

decodeDataObject

Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 161). Subclasses must override this method.

```
- (NSData *)decodeDataObject
```

Discussion

The implementation of your overriding method must match the implementation of your `encodeDataObject:` (page 161) method. For example, a typical `encodeDataObject:` (page 161) method encodes the number of bytes of data followed by the bytes themselves. Your override of this method must read the number of bytes, create an NSData object of the appropriate size, and decode the bytes into the new NSData object. Your overriding method should return an autoreleased NSData object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSCoder.h`

decodeDoubleForKey:

Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 162) or `encodeDouble:forKey:` (page 161) and associated with the string *key*.

```
- (double)decodeDoubleForKey:(NSString *)key
```

Discussion

Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeFloatForKey:

Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 162) or `encodeDouble:forKey:` (page 161) and associated with the string *key*.

```
- (float)decodeFloatForKey:(NSString *)key
```

Discussion
If the value was encoded as a `double`, the extra precision is lost. Also, if the encoded real value does not fit into a `float`, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeInt32ForKey:

Decodes and returns a 32-bit integer value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.

```
- (int32_t)decodeInt32ForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into a 32-bit integer, the method raises an `NSRangeException`. Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeInt64ForKey:

Decodes and returns a 64-bit integer value that was previously encoded with `encodeInt:forKey:` (page 163), `encodeInteger:forKey:` (page 163), `encodeInt32:forKey:` (page 162), or `encodeInt64:forKey:` (page 162) and associated with the string *key*.

```
- (int64_t)decodeInt64ForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeIntegerForKey:

Decodes and returns an NSInteger value that was previously encoded with encodeInt:forKey: (page 163), encodeInteger:forKey: (page 163), encodeInt32:forKey: (page 162), or encodeInt64:forKey: (page 162) and associated with the string *key*.

```
- (NSInteger)decodeIntegerForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into the NSInteger size, the method raises an NSRangeException. Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeIntForKey:

Decodes and returns an int value that was previously encoded with encodeInt:forKey: (page 163), encodeInteger:forKey: (page 163), encodeInt32:forKey: (page 162), or encodeInt64:forKey: (page 162) and associated with the string *key*.

```
- (int)decodeIntForKey:(NSString *)key
```

Discussion
If the encoded integer does not fit into the default integer size, the method raises an NSRangeException. Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSCoder.h

decodeObject

Decodes an Objective-C object that was previously encoded with any of the encode...Object: methods.

```
- (id)decodeObject
```

Discussion
NSCoder's implementation invokes decodeValueOfObjCType:at: (page 156) to decode the object data.

Subclasses may need to override this method if they override any of the corresponding `encode...Object:` methods. For example, if an object was encoded conditionally using the `encodeConditionalObject:` (page 160) method, this method needs to check whether the object had actually been encoded.

The implementation for the concrete subclass `NSUnarchiver` returns an object that is retained by the unarchiver and is released when the unarchiver is deallocated. Therefore, you must retain the returned object before releasing the unarchiver. `NSKeyedUnarchiver`'s implementation, however, returns an autoreleased object, so its life is the same as the current autorelease pool instead of the keyed unarchiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeBycopyObject:` (page 158)
- `encodeByrefObject:` (page 159)
- `encodeObject:` (page 164)

Declared In
`NSCoder.h`

decodeObjectForKey:

Decodes and returns an autoreleased Objective-C object that was previously encoded with `encodeObject:forKey:` (page 164) or `encodeConditionalObject:forKey:` (page 161) and associated with the string *key*.

```
- (id)decodeObjectForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSCoder.h`

decodeValueOfObjCType:at:

Decodes a single value, whose Objective-C type is given by *valueType*.

```
- (void)decodeValueOfObjCType:(const char *)valueType at:(void *)data
```

Discussion
valueType must contain exactly one type code, and the buffer specified by *data* must be large enough to hold the value corresponding to that type code. For information on creating an Objective-C type code suitable for *valueType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Subclasses must override this method and provide an implementation to decode the value. In your overriding implementation, decode the value into the buffer beginning at *data*. If your overriding method is capable of decoding an Objective-C object, your method must also retain that object. Clients of this method are then responsible for releasing the object.

This method matches an `encodeValueOfObjCType:at:` (page 165) message used during encoding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeArrayOfObjCType:count:at:` (page 151)
- `decodeValuesOfObjCTypes:` (page 157)
- `decodeObject` (page 155)

Declared In
`NSCoder.h`

decodeValuesOfObjCTypes:

Decodes a series of potentially different Objective-C types.

```
- (void)decodeValuesOfObjCTypes:(const char *)valueTypes, ...
```

Discussion
valueTypes is a single string containing any number of type codes. The variable arguments to this method consist of one or more pointer arguments, each of which specifies the buffer in which to place a single decoded value. For each type code in *valueTypes*, you must specify a corresponding pointer argument whose buffer is large enough to hold the decoded value. If you use this method to decode Objective-C objects, you are responsible for releasing them.

This method matches an `encodeValuesOfObjCTypes:` (page 165) message used during encoding.

`NSCoder`'s implementation invokes `decodeValueOfObjCType:at:` (page 156) to decode individual types. Subclasses that implement the `decodeValueOfObjCType:at:` (page 156) method do not need to override this method.

For information on creating Objective-C type codes suitable for *valueTypes*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeArrayOfObjCType:count:at:` (page 151)

Declared In
`NSCoder.h`

encodeArrayOfObjCType:count:at:

Encodes an array of *count* items, whose Objective-C type is given by *itemType*.

```
- (void)encodeArrayOfObjCType:(const char *)itemType count:(NSUInteger)count
    at:(const void *)address
```

Discussion
The values are encoded from the buffer beginning at *address*. *itemType* must contain exactly one type code. NSCoder's implementation invokes encodeValueOfObjCType:at: (page 165) to encode the entire array of items. Subclasses that implement the encodeValueOfObjCType:at: (page 165) method do not need to override this method.

This method must be matched by a subsequent decodeArrayOfObjCType:count:at: (page 151) message.

For information on creating an Objective-C type code suitable for *itemType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C Programming Language*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- encodeValueOfObjCType:at: (page 165)
- encodeValuesOfObjCTypes: (page 165)
- encodeBytes:length: (page 159)

Declared In
NSCoder.h

encodeBool:forKey:

Encodes *boolv* and associates it with the string *key*.

```
- (void)encodeBool:(BOOL)boolv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decodeBoolForKey: (page 152)

Declared In
NSCoder.h

encodeBycopyObject:

Can be overridden by subclasses to encode *object* so that a copy, rather than a proxy, is created upon decoding.

```
- (void)encodeBycopyObject:(id)object
```

Discussion
NSCoder's implementation simply invokes encodeObject: (page 164).

This method must be matched by a corresponding `decodeObject` (page 155) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeRootObject:` (page 164)
- `encodeConditionalObject:` (page 160)
- `encodeByrefObject:` (page 159)

Declared In
`NSCoder.h`

encodeByrefObject:

Can be overridden by subclasses to encode *object* so that a proxy, rather than a copy, is created upon decoding.

```
- (void)encodeByrefObject:(id)object
```

Discussion
`NSCoder`'s implementation simply invokes `encodeObject:` (page 164).

This method must be matched by a corresponding `decodeObject` (page 155) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeBycopyObject:` (page 158)

Declared In
`NSCoder.h`

encodeBytes:length:

Encodes a buffer of data whose types are unspecified.

```
- (void)encodeBytes:(const void *)address length:(NSUInteger)numBytes
```

Discussion
The buffer to be encoded begins at *address*, and its length in bytes is given by *numBytes*.

This method must be matched by a corresponding `decodeBytesWithReturnedLength:` (page 153) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeArrayOfObjCType:count:at:` (page 157)

Declared In
NSCoder.h

encodeBytes:length:forKey:

Encodes a buffer of data, *bytesp*, whose length is specified by *lenv*, and associates it with the string *key*.

```
- (void)encodeBytes:(const uint8_t *)bytesp length:(NSUInteger)lenv forKey:(NSString
    *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decodeBytesForKey:returnedLength: (page 152)

Declared In
NSCoder.h

encodeConditionalObject:

Can be overridden by subclasses to conditionally encode *object*, preserving common references to that object.

```
- (void)encodeConditionalObject:(id)object
```

Discussion
In the overriding method, *object* should be encoded only if it's unconditionally encoded elsewhere (with any other encode...Object: method).

This method must be matched by a subsequent decodeObject (page 155) message. Upon decoding, if *object* was never encoded unconditionally, decodeObject returns nil in place of *object*. However, if *object* was encoded unconditionally, all references to *object* must be resolved.

NSCoder's implementation simply invokes encodeObject: (page 164).

Availability
Available in iPhone OS 2.0 and later.

See Also
- encodeRootObject: (page 164)
- encodeObject: (page 164)
- encodeBycopyObject: (page 158)
- encodeConditionalObject: (NSArchiver)

Declared In
NSCoder.h

encodeConditionalObject:forKey:

Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObject:forKey:` (page 164).

```
- (void)encodeConditionalObject:(id)objv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they support keyed coding.

The encoded object is decoded with the `decodeObjectForKey:` (page 156) method. If *objv* was never encoded unconditionally, `decodeObjectForKey:` (page 156) returns `nil` in place of *objv*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSCoder.h`

encodeDataObject:

Encodes a given `NSData` object.

```
- (void)encodeDataObject:(NSData *)data
```

Discussion
Subclasses must override this method.

This method must be matched by a subsequent `decodeDataObject` (page 153) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeObject:` (page 164)

Declared In
`NSCoder.h`

encodeDouble:forKey:

Encodes *realv* and associates it with the string *key*.

```
- (void)encodeDouble:(double)realv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeDoubleForKey:` (page 153)
- `decodeFloatForKey:` (page 154)

Declared In
NSCoder.h

encodeFloat:forKey:

Encodes *realv* and associates it with the string *key*.

```
- (void)encodeFloat:(float)realv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decodeFloatForKey: (page 154)
- decodeDoubleForKey: (page 153)

Declared In
NSCoder.h

encodeInt32:forKey:

Encodes the 32-bit integer *intv* and associates it with the string *key*.

```
- (void)encodeInt32:(int32_t)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decodeIntForKey: (page 155)
- decodeIntegerForKey: (page 155)
- decodeInt32ForKey: (page 154)
- decodeInt64ForKey: (page 154)

Declared In
NSCoder.h

encodeInt64:forKey:

Encodes the 64-bit integer *intv* and associates it with the string *key*.

```
- (void)encodeInt64:(int64_t)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeIntForKey:` (page 155)
- `decodeIntegerForKey:` (page 155)
- `decodeInt32ForKey:` (page 154)
- `decodeInt64ForKey:` (page 154)

Declared In
`NSCoder.h`

encodeInt:forKey:

Encodes *intv* and associates it with the string *key*.

```
- (void)encodeInt:(int)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeIntForKey:` (page 155)
- `decodeIntegerForKey:` (page 155)
- `decodeInt32ForKey:` (page 154)
- `decodeInt64ForKey:` (page 154)

Declared In
`NSCoder.h`

encodeInteger:forKey:

Encodes a given `NSInteger` and associates it with a given key.

```
- (void)encodeInteger:(NSInteger)intv forKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decodeIntForKey:` (page 155)
- `decodeIntegerForKey:` (page 155)
- `decodeInt32ForKey:` (page 154)
- `decodeInt64ForKey:` (page 154)

Declared In
NSCoder.h

encodeObject:

Encodes *object*.

```
- (void)encodeObject:(id)object
```

Discussion
NSCoder's implementation simply invokes encodeValueOfObjCType:at: (page 165) to encode *object*. Subclasses can override this method to encode a reference to *object* instead of *object* itself. For example, NSArchiver detects duplicate objects and encodes a reference to the original object rather than encode the same object twice.

This method must be matched by a subsequent decodeObject (page 155) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- encodeRootObject: (page 164)
- encodeConditionalObject: (page 160)
- encodeBycopyObject: (page 158)

Declared In
NSCoder.h

encodeObject:forKey:

Encodes the object *objv* and associates it with the string *key*.

```
- (void)encodeObject:(id)objv forKey:(NSString *)key
```

Discussion
Subclasses must override this method to identify multiple encodings of *objv* and encode a reference to *objv* instead. For example, NSKeyedArchiver detects duplicate objects and encodes a reference to the original object rather than encode the same object twice.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decodeObjectForKey: (page 156)

Declared In
NSCoder.h

encodeRootObject:

Can be overridden by subclasses to encode an interconnected group of Objective-C objects, starting with *rootObject*.

```
- (void)encodeRootObject:(id)rootObject
```

Discussion
`NSCoder`'s implementation simply invokes `encodeObject:` (page 164).

This method must be matched by a subsequent `decodeObject` (page 155) message.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeObject:` (page 164)
- `encodeConditionalObject:` (page 160)
- `encodeBycopyObject:` (page 158)
- `encodeRootObject:` (`NSArchiver`)

Declared In
`NSCoder.h`

encodeValueOfObjCType:at:

Must be overridden by subclasses to encode a single value residing at *address*, whose Objective-C type is given by *valueType*.

```
- (void)encodeValueOfObjCType:(const char *)valueType at:(const void *)address
```

Discussion
valueType must contain exactly one type code.

This method must be matched by a subsequent `decodeValueOfObjCType:at:` (page 156) message.

For information on creating an Objective-C type code suitable for *valueType*, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `encodeArrayOfObjCType:count:at:` (page 157)
- `encodeValuesOfObjCTypes:` (page 165)

Declared In
`NSCoder.h`

encodeValuesOfObjCTypes:

Encodes a series of values of potentially differing Objective-C types.

```
- (void)encodeValuesOfObjCTypes:(const char *)valueTypes, ...
```

Discussion

valueTypes is a single string containing any number of type codes. The variable arguments to this method consist of one or more pointer arguments, each of which specifies a buffer containing the value to be encoded. For each type code in *valueTypes*, you must specify a corresponding pointer argument.

This method must be matched by a subsequent `decodeValuesOfObjCTypes:` (page 157) message.

`NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 165) to encode individual types. Subclasses that implement the `encodeValueOfObjCType:at:` (page 165) method do not need to override this method. However, subclasses that provide a more efficient approach for encoding a series of values may override this method to implement that approach.

For information on creating Objective-C type codes suitable for valueTypes, see the "Type Encodings" section in "The Objective-C Runtime System" chapter of *The Objective-C 2.0 Programming Language*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `encodeArrayOfObjCType:count:at:` (page 157)
- `encodeValueOfObjCType:at:` (page 165)

Declared In

`NSCoder.h`

objectZone

Returns the memory zone used to allocate decoded objects.

```
- (NSZone *)objectZone
```

Discussion

`NSCoder`'s implementation simply returns the default memory zone, as given by `NSDefaultMallocZone()`.

Subclasses must override this method and the `setObjectZone:` (page 166) method to allow objects to be decoded into a zone other than the default zone. In its overriding implementation of this method, your subclass should return the current memory zone (if one has been set) or the default zone (if no other zone has been set).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSCoder.h`

setObjectZone:

`NSCoder`'s implementation of this method does nothing.

```
- (void)setObjectZone:(NSZone *)zone
```

Discussion
Can be overridden by subclasses to set the memory zone used to allocate decoded objects.

Subclasses must override this method and `objectZone` (page 166) to allow objects to be decoded into a zone other than the default zone. In its overriding implementation of this method, your subclass should store a reference to the current memory zone.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSCoder.h`

systemVersion
During encoding, this method should return the system version currently in effect.

```
- (unsigned)systemVersion
```

Discussion
During decoding, this method should return the version that was in effect when the data was encoded.

By default, this method returns the current system version, which is appropriate for encoding but not for decoding. Subclasses that implement decoding must override this method to return the system version of the data being decoded.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSCoder.h`

versionForClassName:
Returns the version in effect for the class with a given name.

```
- (NSInteger)versionForClassName:(NSString *)className
```

Return Value
The version in effect for the class named *`className`* or `NSNotFound` if no class named *`className`* exists.

Discussion
When encoding, this method returns the current version number of the class. When decoding, this method returns the version number of the class being decoded. Subclasses must override this method.

Special Considerations
The version number applies to `NSArchiver`/`NSUnarchiver`, but not to `NSKeyedArchiver`/`NSKeyedUnarchiver`. A keyed archiver does not encode class version numbers.

Availability
Available in iPhone OS 2.0 and later.

See Also

+ setVersion: (page 795) (NSObject)
+ version (page 796) (NSObject)

Declared In

NSCoder.h

NSCondition Class Reference

Inherits from:	NSObject
Conforms to:	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLock.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSCondition` class implements a condition variable whose semantics follow those used for POSIX-style conditions. A condition object acts as both a lock and a checkpoint in a given thread. The lock protects your code while it tests the condition and performs the task triggered by the condition. The checkpoint behavior requires that the condition be true before the thread proceeds with its task. While the condition is not true, the thread blocks. It remains blocked until another thread signals the condition object.

The semantics for using an `NSCondition` object are as follows:

1. Lock the condition object.

2. Test a boolean predicate. (This predicate is a boolean flag or other variable in your code that indicates whether it is safe to perform the task protected by the condition.)
3. If the boolean predicate is false, call the condition object's `wait` or `waitUntilDate:` method to block the thread. Upon returning from these methods, go to step 2 to retest your boolean predicate. (Continue waiting and retesting the predicate until it is true.)
4. If the boolean predicate is true, perform the task.
5. Optionally update any predicates (or signal any conditions) affected by your task.
6. When your task is done, unlock the condition object.

The pseudocode for performing the preceding steps would therefore look something like the following:

```
lock the condition
while (!(boolean_predicate)) {
    wait on condition
}
do protected work
(optionally, signal or broadcast the condition again or change a predicate value)
unlock the condition
```

Whenever you use a condition object, the first step is to lock the condition. Locking the condition ensures that your predicate and task code are protected from interference by other threads using the same condition. Once you have completed your task, you can set other predicates or signal other conditions based on the needs of your code. You should always set predicates and signal conditions while holding the condition object's lock.

When a thread waits on a condition, the condition object unlocks its lock and blocks the thread. When the condition is signaled, the system wakes up the thread. The condition object then reacquires its lock before returning from the `wait` or `waitUntilDate:` method. Thus, from the point of view of the thread, it is as if it always held the lock.

A boolean predicate is an important part of the semantics of using conditions because of the way signaling works. Signaling a condition does not guarantee that the condition itself is true. There are timing issues involved in signaling that may cause false signals to appear. Using a predicate ensures that these spurious signals do not cause you to perform work before it is safe to do so. The predicate itself is simply a flag or other variable in your code that you test in order to acquire a Boolean result.

For more information on how to use conditions, see Using POSIX Thread Locks in *Threading Programming Guide*.

Tasks

Waiting for the Lock

`- wait` (page 172)
Blocks the current thread until the condition is signaled.

`- waitUntilDate:` (page 173)
Blocks the current thread until the condition is signaled or the specified time limit is reached.

Signaling Waiting Threads

- `signal` (page 172)
  Signals the condition, waking up one thread waiting on it.
- `broadcast` (page 171)
  Signals the condition, waking up all threads waiting on it.

Accessor Methods

- `setName:` (page 172)
  Assigns a name to the receiver.
- `name` (page 171)
  Returns the name associated with the receiver.

Instance Methods

broadcast

Signals the condition, waking up all threads waiting on it.

```
- (void)broadcast
```

Discussion
If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLock.h`

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setName:` (page 172)

Declared In
NSLock.h

setName:

Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a condition object within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- name (page 171)

Declared In
NSLock.h

signal

Signals the condition, waking up one thread waiting on it.

```
- (void)signal
```

Discussion
You use this method to wake up one thread that is waiting on the condition. You may call this method multiple times to wake up multiple threads. If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSLock.h

wait

Blocks the current thread until the condition is signaled.

```
- (void)wait
```

Discussion
You must lock the receiver prior to calling this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lock (page 1298) (NSLocking)

Declared In
NSLock.h

waitUntilDate:

Blocks the current thread until the condition is signaled or the specified time limit is reached.

```
- (BOOL)waitUntilDate:(NSDate *)limit
```

Parameters

limit

The time at which to wake up the thread if the condition has not been signaled.

Return Value
YES if the condition was signaled; otherwise, NO if the time limit was reached.

Discussion
You must lock the receiver prior to calling this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lock (page 1298) (NSLocking)

Declared In
NSLock.h

NSConditionLock Class Reference

Inherits from:	NSObject
Conforms to:	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLock.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSConditionLock` class defines objects whose locks can be associated with specific, user-defined conditions. Using an `NSConditionLock` object, you can ensure that a thread can acquire a lock only if a certain condition is met. Once it has acquired the lock and executed the critical section of code, the thread can relinquish the lock and set the associated condition to something new. The conditions themselves are arbitrary: you define them as needed for your application.

Adopted Protocols

NSLocking
- lock (page 1298)
- unlock (page 1298)

Tasks

Initializing an NSConditionLock Object

- initWithCondition: (page 177)
  Initializes a newly allocated NSConditionLock object and sets its condition.

Returning the Condition

- condition (page 177)
  Returns the condition associated with the receiver.

Acquiring and Releasing a Lock

- lockBeforeDate: (page 177)
  Attempts to acquire a lock before a specified moment in time.
- lockWhenCondition: (page 178)
  Attempts to acquire a lock.
- lockWhenCondition:beforeDate: (page 178)
  Attempts to acquire a lock before a specified moment in time.
- tryLock (page 180)
  Attempts to acquire a lock without regard to the receiver's condition.
- tryLockWhenCondition: (page 180)
  Attempts to acquire a lock if the receiver's condition is equal to the specified condition.
- unlockWithCondition: (page 180)
  Relinquishes the lock and sets the receiver's condition.

Accessor Methods

- setName: (page 179)
  Assigns a name to the receiver.
- name (page 179)
  Returns the name associated with the receiver.

Instance Methods

condition

Returns the condition associated with the receiver.

```
- (NSInteger)condition
```

Return Value

The condition associated with the receiver. If no condition has been set, returns 0.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSLock.h

initWithCondition:

Initializes a newly allocated `NSConditionLock` object and sets its condition.

```
- (id)initWithCondition:(NSInteger)condition
```

Parameters

condition

The user-defined condition for the lock. The value of *condition* is user-defined; see the class description for more information.

Return Value

An initialized condition lock object; may be different than the original receiver.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSLock.h

lockBeforeDate:

Attempts to acquire a lock before a specified moment in time.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit

The date by which the lock must be acquired or the attempt will time out.

Return Value

`YES` if the lock is acquired within the time limit, `NO` otherwise.

Discussion
The condition associated with the receiver isn't taken into account in this operation. This method blocks the thread's execution until the receiver acquires the lock or *limit* is reached.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lockWhenCondition:beforeDate: (page 178)

Declared In
NSLock.h

lockWhenCondition:
Attempts to acquire a lock.

```
- (void)lockWhenCondition:(NSInteger)condition
```

Parameters
condition
The condition to match on.

Discussion
The receiver's condition must be equal to *condition* before the locking operation will succeed. This method blocks the thread's execution until the lock can be acquired.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lockWhenCondition:beforeDate: (page 178)
- unlockWithCondition: (page 180)

Declared In
NSLock.h

lockWhenCondition:beforeDate:
Attempts to acquire a lock before a specified moment in time.

```
- (BOOL)lockWhenCondition:(NSInteger)condition beforeDate:(NSDate *)limit
```

Parameters
condition
The condition to match on.

limit
The date by which the lock must be acquired or the attempt will time out.

Return Value
YES if the lock is acquired within the time limit, NO otherwise.

Discussion
The receiver's condition must be equal to *condition* before the locking operation will succeed. This method blocks the thread's execution until the lock can be acquired or *limit* is reached.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lockBeforeDate: (page 177)
- lockWhenCondition: (page 178)

Declared In
NSLock.h

name
Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setName: (page 179)

Declared In
NSLock.h

setName:
Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters
newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a condition lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- name (page 179)

Declared In
NSLock.h

tryLock

Attempts to acquire a lock without regard to the receiver's condition.

```
- (BOOL)tryLock
```

Return Value

`YES` if the lock could be acquired, `NO` otherwise.

Discussion

This method returns immediately.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- tryLockWhenCondition:` (page 180)

Declared In

`NSLock.h`

tryLockWhenCondition:

Attempts to acquire a lock if the receiver's condition is equal to the specified condition.

```
- (BOOL)tryLockWhenCondition:(NSInteger)condition
```

Return Value

`YES` if the lock could be acquired, `NO` otherwise.

Discussion

As part of its implementation, this method invokes `lockWhenCondition:beforeDate:` (page 178). This method returns immediately.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- tryLock` (page 180)

Declared In

`NSLock.h`

unlockWithCondition:

Relinquishes the lock and sets the receiver's condition.

```
- (void)unlockWithCondition:(NSInteger)condition
```

Parameters

condition

The user-defined condition for the lock. The value of *condition* is user-defined; see the class description for more information.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- lockWhenCondition:` (page 178)

Declared In
`NSLock.h`

NSCountedSet Class Reference

Inherits from:	NSMutableSet : NSSet : NSObject
Conforms to:	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSSet.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSCountedSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSCountedSet` provides support for the mathematical concept of a counted set. A counted set, both in its mathematical sense and in the implementation of `NSCountedSet`, is an unordered collection of elements, just as in a regular set, but the elements of the set aren't necessarily distinct. A counted set is also known as a bag.

Each distinct object inserted into an `NSCountedSet` object has a counter associated with it. `NSCountedSet` keeps track of the number of times objects are inserted and requires that objects be removed the same number of times. Thus, there is only one instance of an object in an `NSSet` object even if the object has been added to the set multiple times. The `count` (page 925) method defined by the superclass `NSSet` has special significance; it returns the number of distinct objects, not the total number of times objects are represented in the set. The `NSSet` and `NSMutableSet` classes are provided for static and dynamic sets (respectively) whose elements are distinct.

You add objects to or remove objects from a counted set using the `addObject:` (page 185) and `removeObject:` (page 187) methods. You can traverse elements of an `NSCountedSet` object using the enumerator returned by `objectEnumerator` (page 187). The `countForObject:` (page 185) method returns the number of times a given object has been added to this set.

Tasks

Initializing a Counted Set

- `initWithArray:` (page 185)
  Returns a counted set object initialized with the contents of a given array.
- `initWithSet:` (page 186)
  Returns a counted set object initialized with the contents of a given set.
- `initWithCapacity:` (page 186)
  Returns a counted set object initialized with enough memory to hold a given number of objects.

Adding and Removing Entries

- `addObject:` (page 185)
  Adds a given object to the receiver.
- `removeObject:` (page 187)
  Removes a given object from the receiver.

Examining a Counted Set

- `countForObject:` (page 185)
  Returns the count associated with a given object in the receiver.
- `objectEnumerator` (page 187)
  Returns an enumerator object that lets you access each object in the set once, independent of its count.

Instance Methods

addObject:

Adds a given object to the receiver.

```
- (void)addObject:(id)anObject
```

Parameters

anObject

The object to add to the receiver.

Discussion

If *anObject* is already a member, `addObject:` increments the count associated with the object. If *anObject* is not already a member, it is sent a `retain` (page 1312) message.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSSet.h`

countForObject:

Returns the count associated with a given object in the receiver.

```
- (NSUInteger)countForObject:(id)anObject
```

Parameters

anObject

The object for which to return the count.

Return Value

The count associated with *anObject* in the receiver, which can be thought of as the number of occurrences of *anObject* present in the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- count` (page 925) (`NSSet`)

Declared In

`NSSet.h`

initWithArray:

Returns a counted set object initialized with the contents of a given array.

```
- (id)initWithArray:(NSArray *)anArray
```

Parameters

anArray

An array of objects to add to the new set.

Return Value

An initialized counted set object with the contents of *anArray*. The returned object might be different than the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`initWithArray:` (page 926) (NSSet)

`setWithArray:` (page 920) (NSSet)

Declared In

`NSSet.h`

initWithCapacity:

Returns a counted set object initialized with enough memory to hold a given number of objects.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the new counted set.

Return Value

A counted set object initialized with enough memory to hold *numItems* objects

Discussion

The method is the designated initializer for `NSCountedSet`.

Note that the capacity is simply a hint to help initial memory allocation—the initial count of the object is `0`, and the set still grows and shrinks as you add and remove objects. The hint is typically useful if the set will become large.

Availability

Available in iPhone OS 2.0 and later.

See Also

`initWithCapacity:` (page 620) (NSMutableSet)

`setWithCapacity:` (page 619) (NSMutableSet)

Declared In

`NSSet.h`

initWithSet:

Returns a counted set object initialized with the contents of a given set.

```
- (id)initWithSet:(NSSet *)aSet
```

Parameters

aSet

An set of objects to add to the new set.

Return Value

An initialized counted set object with the contents of *aSet*. The returned object might be different than the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`initWithSet:` (page 927) (NSSet)

`setWithSet:` (page 922) (NSSet)

Declared In

`NSSet.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the set once, independent of its count.

```
- (NSEnumerator *)objectEnumerator
```

Return Value

An enumerator object that lets you access each object in the set once, independent of its count.

Discussion

If you add a given object to the counted set multiple times, an enumeration of the set will produce that object only once.

You shouldn't modify the set during enumeration. If you intend to modify the set, use the `allObjects` (page 923) method to create a "snapshot," then enumerate the snapshot and modify the original set.

Availability

Available in iPhone OS 2.0 and later.

See Also

`nextObject` (page 341) (NSEnumerator)

Declared In

`NSSet.h`

removeObject:

Removes a given object from the receiver.

```
- (void)removeObject:(id)anObject
```

Parameters

anObject

The object to remove from the receiver.

Discussion

If *anObject* is present in the set, decrements the count associated with it. If the count is decremented to 0, *anObject* is removed from the set and sent a `release` (page 1310) message. `removeObject:` does nothing if *anObject* is not present in the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- countForObject:` (page 185)

Declared In

`NSSet.h`

NSData Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guides:	Binary Data Programming Guide for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSData` and its mutable subclass `NSMutableData` provide data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Foundation objects.

NSData creates static data objects, and NSMutableData creates dynamic data objects. NSData and NSMutableData are typically used for data storage and are also useful in Distributed Objects applications, where data contained in data objects can be copied or moved between applications.

Using 32-bit Cocoa, the size of the data is subject to a theoretical 2GB limit (in practice, because memory will be used by other objects this limit will be smaller); using 64-bit Cocoa, the size of the data is subject to a theoretical limit of about 8EB (in practice, the limit should not be a factor).

NSData is "toll-free bridged" with its Core Foundation counterpart, CFData. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an NSData * parameter, you can pass a CFDataRef, and in a function where you see a CFDataRef parameter, you can pass an NSData instance (you cast one type to the other to suppress compiler warnings). This also applies to your concrete subclasses of NSData. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

NSMutableCopying
- mutableCopyWithZone: (page 1300)

Tasks

Creating Data Objects

+ data (page 192)
Creates and returns an empty data object.

+ dataWithBytes:length: (page 192)
Creates and returns a data object containing a given number of bytes copied from a given buffer.

+ dataWithBytesNoCopy:length: (page 193)
Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

+ dataWithBytesNoCopy:length:freeWhenDone: (page 194)
Creates and returns a data object that holds a given number of bytes from a given buffer.

+ dataWithContentsOfFile: (page 194)
Creates and returns a data object by reading every byte from the file specified by a given path.

+ dataWithContentsOfFile:options:error: (page 195)
Creates and returns a data object by reading every byte from the file specified by a given path.

+ dataWithContentsOfMappedFile: (page 195)
Creates and returns a data object from the mapped file specified by *path*.

+ dataWithContentsOfURL: (page 196)
Returns a data object containing the data from the location specified by a given URL.

+ dataWithContentsOfURL:options:error: (page 196)
Creates and returns a data object containing the data from the location specified by *aURL*.

+ dataWithData: (page 197)
Creates and returns a data object containing the contents of another data object.

- initWithBytes:length: (page 200)
Returns a data object initialized by adding to it a given number of bytes of data copied from a given buffer.

- initWithBytesNoCopy:length: (page 200)
Returns a data object initialized by adding to it a given number of bytes of data from a given buffer.

- initWithBytesNoCopy:length:freeWhenDone: (page 201)
Initializes a newly allocated data object by adding to it *length* bytes of data from the buffer *bytes*.

- initWithContentsOfFile: (page 201)
Returns a data object initialized by reading into it the data from the file specified by a given path.

- initWithContentsOfFile:options:error: (page 202)
Returns a data object initialized by reading into it the data from the file specified by a given path.

- initWithContentsOfMappedFile: (page 202)
Returns a data object initialized by reading into it the mapped file specified by a given path.

- initWithContentsOfURL: (page 203)
Initializes a newly allocated data object initialized with the data from the location specified by *aURL*.

- initWithContentsOfURL:options:error: (page 203)
Returns a data object initialized with the data from the location specified by a given URL.

- initWithData: (page 204)
Returns a data object initialized with the contents of another data object.

Accessing Data

- bytes (page 197)
Returns a pointer to the receiver's contents.

- description (page 198)
Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.

- getBytes: (page 198)
Copies a data object's contents into a given buffer.

- getBytes:length: (page 199)
Copies up to a given number of bytes from the start of the receiver's data into a given buffer.

- getBytes:range: (page 199)
Copies into a given buffer the contents from a given range within the bytes in the receiver.

- `subdataWithRange:` (page 205)
  Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by a given range.

Testing Data

- `isEqualToData:` (page 204)
  Compares the receiving data object to *otherData*.
- `length` (page 205)
  Returns the number of bytes contained in the receiver.

Storing Data

- `writeToFile:atomically:` (page 205)
  Writes the bytes in the receiver to the file specified by a given path.
- `writeToFile:options:error:` (page 206)
  Writes the bytes in the receiver to the file specified by a given path.
- `writeToURL:atomically:` (page 206)
  Writes the bytes in the receiver to the location specified by *aURL*.
- `writeToURL:options:error:` (page 207)
  Writes the bytes in the receiver to the location specified by a given URL.

Class Methods

data

Creates and returns an empty data object.

```
+ (id)data
```

Return Value
An empty data object.

Discussion
This method is declared primarily for the use of mutable subclasses of `NSData`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSData.h`

dataWithBytes:length:

Creates and returns a data object containing a given number of bytes copied from a given buffer.

```
+ (id)dataWithBytes:(const void *)bytes length:(NSUInteger)length
```

Parameters

bytes

A buffer containing data for the new object.

length

The number of bytes to copy from *bytes*. This value must not exceed the length of *bytes*.

Return Value

A data object containing *length* bytes copied from the buffer *bytes*. Returns `nil` if the data object could not be created.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dataWithBytesNoCopy:length:` (page 193)

`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 194)

Declared In

`NSData.h`

dataWithBytesNoCopy:length:

Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

```
+ (id)dataWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
```

Parameters

bytes

A buffer containing data for the new object. *bytes* must point to a memory block allocated with `malloc`.

length

The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

Return Value

A data object that holds *length* bytes from the buffer *bytes*. Returns `nil` if the data object could not be created.

Discussion

The returned object takes ownership of the *bytes* pointer and frees it on deallocation. Therefore, *bytes* must point to a memory block allocated with `malloc`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dataWithBytes:length:` (page 192)

`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 194)

Declared In

`NSData.h`

dataWithBytesNoCopy:length:freeWhenDone:

Creates and returns a data object that holds a given number of bytes from a given buffer.

```
+ (id)dataWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    freeWhenDone:(BOOL)freeWhenDone
```

Parameters

bytes

A buffer containing data for the new object. If *freeWhenDone* is YES, *bytes* must point to a memory block allocated with malloc.

length

The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

freeWhenDone

If YES, the returned object takes ownership of the *bytes* pointer and frees it on deallocation.

Return Value

A data object that holds *length* bytes from the buffer *bytes*. Returns nil if the data object could not be created.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ dataWithBytes:length: (page 192)
+ dataWithBytesNoCopy:length: (page 193)

Declared In

NSData.h

dataWithContentsOfFile:

Creates and returns a data object by reading every byte from the file specified by a given path.

```
+ (id)dataWithContentsOfFile:(NSString *)path
```

Parameters

path

The absolute path of the file from which to read data.

Return Value

A data object by reading every byte from the file specified by *path*. Returns nil if the data object could not be created.

Discussion

This method is equivalent to dataWithContentsOfFile:options:error: (page 195) with no options. If you need to know what was the reason for failure, use dataWithContentsOfFile:options:error: (page 195).

A sample using this method can be found in Working With Binary Data.

Availability

Available in iPhone OS 2.0 and later.

See Also
+ dataWithContentsOfFile:options:error: (page 195)
+ dataWithContentsOfMappedFile: (page 195)

Declared In
NSData.h

dataWithContentsOfFile:options:error:

Creates and returns a data object by reading every byte from the file specified by a given path.

```
+ (id)dataWithContentsOfFile:(NSString *)path options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters
path

The absolute path of the file from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 208).

errorPtr

If an error occurs, upon return contains an NSError object that describes the problem.

Return Value
A data object by reading every byte from the file specified by *path*. Returns nil if the data object could not be created.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSData.h

dataWithContentsOfMappedFile:

Creates and returns a data object from the mapped file specified by *path*.

```
+ (id)dataWithContentsOfMappedFile:(NSString *)path
```

Parameters
path

The absolute path of the file from which to read data.

Return Value
A data object from the mapped file specified by *path*. Returns nil if the data object could not be created.

Discussion
Because of file mapping restrictions, this method should only be used if the file is guaranteed to exist for the duration of the data object's existence. It is generally safer to use the dataWithContentsOfFile: (page 194) method.

This methods assumes mapped files are available from the underlying operating system. A mapped file uses virtual memory techniques to avoid copying pages of the file into memory until they are actually needed.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dataWithContentsOfFile:` (page 194)

Declared In
`NSData.h`

dataWithContentsOfURL:

Returns a data object containing the data from the location specified by a given URL.

```
+ (id)dataWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The URL from which to read data.

Return Value
A data object containing the data from the location specified by *aURL*. Returns `nil` if the data object could not be created.

Discussion
If you need to know what was the reason for failure, use `dataWithContentsOfURL:options:error:` (page 196).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dataWithContentsOfURL:options:error:` (page 196)
`- initWithContentsOfURL:` (page 203)

Declared In
`NSData.h`

dataWithContentsOfURL:options:error:

Creates and returns a data object containing the data from the location specified by *aURL*.

```
+ (id)dataWithContentsOfURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

aURL

The URL from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 208).

errorPtr

If there is an error reading in the data, upon return contains an `NSError` object that describes the problem.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithContentsOfURL:` (page 203)

Declared In
`NSData.h`

dataWithData:

Creates and returns a data object containing the contents of another data object.

```
+ (id)dataWithData:(NSData *)aData
```

Parameters

aData

A data object.

Return Value
A data object containing the contents of *aData*. Returns `nil` if the data object could not be created.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithData:` (page 204)

Declared In
`NSData.h`

Instance Methods

bytes

Returns a pointer to the receiver's contents.

```
- (const void *)bytes
```

Return Value
A read-only pointer to the receiver's contents.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `description` (page 198)
- `getBytes:` (page 198)
- `getBytes:length:` (page 199)
- `getBytes:range:` (page 199)

Declared In
`NSData.h`

description

Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.

```
- (NSString *)description
```

Return Value
An `NSString` object that contains a hexadecimal representation of the receiver's contents in `NSData` property list format.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `bytes` (page 197)
- `getBytes:` (page 198)
- `getBytes:length:` (page 199)
- `getBytes:range:` (page 199)

Declared In
`NSData.h`

getBytes:

Copies a data object's contents into a given buffer.

```
- (void)getBytes:(void *)buffer
```

Parameters
buffer

A buffer into which to copy the receiver's data. The buffer must be at least `length` (page 205) bytes.

Discussion
You can see a sample using this method in Working With Binary Data.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `bytes` (page 197)
- `description` (page 198)
- `getBytes:length:` (page 199)

- getBytes:range: (page 199)

Declared In
NSData.h

getBytes:length:

Copies up to a given number of bytes from the start of the receiver's data into a given buffer.

```
- (void)getBytes:(void *)buffer length:(NSUInteger)length
```

Parameters

buffer
A buffer into which to copy data.

length
The number of bytes from the start of the receiver's data to copy to *buffer*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- bytes (page 197)
- description (page 198)
- getBytes: (page 198)
- getBytes:range: (page 199)

Declared In
NSData.h

getBytes:range:

Copies into a given buffer the contents from a given range within the bytes in the receiver.

```
- (void)getBytes:(void *)buffer range:(NSRange)range
```

Parameters

buffer
A buffer into which to copy data.

range
The range of bytes in the receiver's data to copy to *buffer*. The range must lie within the range of bytes of the receiver's data.

Discussion
If *range* isn't within the receiver's range of bytes, an NSRangeException is raised.

Availability
Available in iPhone OS 2.0 and later.

See Also
- bytes (page 197)
- description (page 198)
- getBytes: (page 198)

- getBytes:length: (page 199)

Declared In
NSData.h

initWithBytes:length:

Returns a data object initialized by adding to it a given number of bytes of data copied from a given buffer.

- (id)initWithBytes:(const void *)*bytes* length:(NSUInteger)*length*

Discussion
A data object initialized by adding to it *length* bytes of data copied from the buffer *bytes*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithBytes:length: (page 192)
- initWithBytesNoCopy:length: (page 200)
- initWithBytesNoCopy:length:freeWhenDone: (page 201)

Declared In
NSData.h

initWithBytesNoCopy:length:

Returns a data object initialized by adding to it a given number of bytes of data from a given buffer.

- (id)initWithBytesNoCopy:(void *)*bytes* length:(NSUInteger)*length*

Parameters
bytes
A buffer containing data for the new object. *bytes* must point to a memory block allocated with malloc.

length
The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

Return Value
A data object initialized by adding to it *length* bytes of data from the buffer *bytes*. The returned object might be different than the original receiver.

Discussion
The returned object takes ownership of the *bytes* pointer and frees it on deallocation. Therefore, *bytes* must point to a memory block allocated with malloc.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithBytes:length: (page 192)

- initWithBytes:length: (page 200)
- initWithBytesNoCopy:length:freeWhenDone: (page 201)

Declared In
NSData.h

initWithBytesNoCopy:length:freeWhenDone:

Initializes a newly allocated data object by adding to it *length* bytes of data from the buffer *bytes*.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Parameters

bytes
A buffer containing data for the new object. If *flag* is YES, *bytes* must point to a memory block allocated with malloc.

length
The number of bytes to hold from *bytes*. This value must not exceed the length of *bytes*.

flag
If YES, the returned object takes ownership of the *bytes* pointer and frees it on deallocation.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithBytesNoCopy:length:freeWhenDone: (page 194)
- initWithBytes:length: (page 200)
- initWithBytesNoCopy:length: (page 200)

Declared In
NSData.h

initWithContentsOfFile:

Returns a data object initialized by reading into it the data from the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)path
```

Parameters

path
The absolute path of the file from which to read data.

Return Value
A data object initialized by reading into it the data from the file specified by *path*. The returned object might be different than the original receiver.

Discussion
This method is equivalent to initWithContentsOfFile:options:error: (page 202) with no options.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithContentsOfFile: (page 194)
- initWithContentsOfMappedFile: (page 202)

Declared In
NSData.h

initWithContentsOfFile:options:error:

Returns a data object initialized by reading into it the data from the file specified by a given path.

```
- (id)initWithContentsOfFile:(NSString *)path options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters

path
The absolute path of the file from which to read data.

mask
A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 208).

errorPtr
If an error occurs, upon return contains an NSError object that describes the problem.

Return Value
A data object initialized by reading into it the data from the file specified by *path*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithContentsOfFile:options:error: (page 195)

Declared In
NSData.h

initWithContentsOfMappedFile:

Returns a data object initialized by reading into it the mapped file specified by a given path.

```
- (id)initWithContentsOfMappedFile:(NSString *)path
```

Parameters

path
The absolute path of the file from which to read data.

Return Value
A data object initialized by reading into it the mapped file specified by *path*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithContentsOfMappedFile: (page 195)
- initWithContentsOfFile: (page 201)

Declared In
NSData.h

initWithContentsOfURL:

Initializes a newly allocated data object initialized with the data from the location specified by *aURL*.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

The URL from which to read data

Return Value
An NSData object initialized with the data from the location specified by *aURL*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithContentsOfURL: (page 196)

Declared In
NSData.h

initWithContentsOfURL:options:error:

Returns a data object initialized with the data from the location specified by a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

aURL

The URL from which to read data.

mask

A mask that specifies options for reading the data. Constant components are described in "Options for NSData Reading Methods" (page 208).

errorPtr

If there is an error reading in the data, upon return contains an NSError object that describes the problem.

Return Value
A data object initialized with the data from the location specified by *aURL*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dataWithContentsOfURL:options:error:` (page 196)

Declared In
`NSData.h`

initWithData:

Returns a data object initialized with the contents of another data object.

```
- (id)initWithData:(NSData *)data
```

Parameters

data
A data object.

Return Value
A data object initialized with the contents *data*. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dataWithData:` (page 197)

Declared In
`NSData.h`

isEqualToData:

Compares the receiving data object to *otherData*.

```
- (BOOL)isEqualToData:(NSData *)otherData
```

Parameters

otherData
The data object with which to compare the receiver.

Return Value
`YES` if the contents of *otherData* are equal to the contents of the receiver, otherwise `NO`.

Discussion
Two data objects are equal if they hold the same number of bytes, and if the bytes at the same position in the objects are the same.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSData.h`

length

Returns the number of bytes contained in the receiver.

```
- (NSUInteger)length
```

Return Value
The number of bytes contained in the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSData.h

subdataWithRange:

Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by a given range.

```
- (NSData *)subdataWithRange:(NSRange)range
```

Parameters
range

The range in the receiver from which to copy bytes. The range must not exceed the bounds of the receiver.

Return Value
A data object containing a copy of the receiver's bytes that fall within the limits specified by *range*.

Discussion
If *range* isn't within the receiver's range of bytes, an NSRangeException is raised.

A sample using this method can be found in Working With Binary Data.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSData.h

writeToFile:atomically:

Writes the bytes in the receiver to the file specified by a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters
path

The location to which to write the receiver's bytes. If *path* contains a tilde (~) character, you must expand it with stringByExpandingTildeInPath (page 1031) before invoking this method.

atomically

If YES, the data is written to a backup file, and then—assuming no errors occur—the backup file is renamed to the name specified by *path*; otherwise, the data is written directly to *path*.

Return Value
YES if the operation succeeds, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- writeToURL:atomically: (page 206)

Declared In
NSData.h

writeToFile:options:error:

Writes the bytes in the receiver to the file specified by a given path.

```
- (BOOL)writeToFile:(NSString *)path options:(NSUInteger)mask error:(NSError
    **)errorPtr
```

Parameters

path

The location to which to write the receiver's bytes.

mask

A mask that specifies options for writing the data. Constant components are described in "Options for NSData Writing Methods" (page 208).

errorPtr

If there is an error writing out the data, upon return contains an NSError object that describes the problem.

Return Value
YES if the operation succeeds, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- writeToURL:options:error: (page 207)

Declared In
NSData.h

writeToURL:atomically:

Writes the bytes in the receiver to the location specified by *aURL*.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)atomically
```

Parameters

aURL

The location to which to write the receiver's bytes. Only file:// URLs are supported.

atomically

If YES, the data is written to a backup location, and then—assuming no errors occur—the backup location is renamed to the name specified by *aURL*; otherwise, the data is written directly to *aURL*. *atomically* is ignored if *aURL* is not of a type the supports atomic writes.

Return Value

YES if the operation succeeds, otherwise NO.

Discussion

Since at present only file:// URLs are supported, there is no difference between this method and writeToFile:atomically: (page 205), except for the type of the first argument.

Availability

Available in iPhone OS 2.0 and later.

See Also

- writeToFile:atomically: (page 205)

Declared In

NSData.h

writeToURL:options:error:

Writes the bytes in the receiver to the location specified by a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL options:(NSUInteger)mask error:(NSError **)errorPtr
```

Parameters

aURL

The location to which to write the receiver's bytes.

mask

A mask that specifies options for writing the data. Constant components are described in "Options for NSData Writing Methods" (page 208).

errorPtr

If there is an error writing out the data, upon return contains an NSError object that describes the problem.

Return Value

YES if the operation succeeds, otherwise NO.

Discussion

Since at present only file:// URLs are supported, there is no difference between this method and writeToFile:options:error: (page 206), except for the type of the first argument.

Availability

Available in iPhone OS 2.0 and later.

See Also

- writeToFile:options:error: (page 206)

Declared In

NSData.h

Constants

Options for NSData Reading Methods

Options for methods used to read `NSData` objects.

```
enum {
    NSMappedRead = 1,
    NSUncachedRead = 2
};
```

Constants

`NSMappedRead`

A hint indicating the file should be mapped into virtual memory, if possible.

Available in iPhone OS 2.0 and later.

Declared in `NSData.h`

`NSUncachedRead`

A hint indicating the file should not be stored in the file-system caches.

For data being read once and discarded, this option can improve performance.

Available in iPhone OS 2.0 and later.

Declared in `NSData.h`

Declared In

`NSData.h`

Options for NSData Writing Methods

Options for methods used to write `NSData` objects.

```
enum {
    NSAtomicWrite = 1
};
```

Constants

`NSAtomicWrite`

A hint to use an auxiliary file when saving data and then exchange the files.

Available in iPhone OS 2.0 and later.

Declared in `NSData.h`

Declared In

`NSData.h`

NSDate Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDate.h Foundation/NSCalendarDate.h
Companion guides:	Date and Time Programming Guide for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSDate` objects represent a single point in time. `NSDate` is a class cluster; its 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 instances not 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`. Generally, you instantiate a suitable date object by invoking one of the `date...` class methods.

NSDate is an abstract class that provides behavior for creating dates, comparing dates, representing dates, computing intervals, and similar functionality. NSDate 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 an invariant point in time. This class is designed to provide the foundation for arbitrary calendrical representations.

The sole primitive method of NSDate, timeIntervalSinceReferenceDate (page 223), provides the basis for all the other methods in the NSDate interface. This method returns a time value relative to an absolute reference date—the first instant of 1 January 2001, GMT.

NSDate provides several methods to interpret and to create string representations of dates (for example, dateWithNaturalLanguageString:locale: and descriptionWithLocale:). In general, on Mac OS X v10.4 and later you should use an instance of NSDateFormatter to parse and generate strings using the methods dateFromString: (page 244) and stringFromDate: (page 268)—see NSDateFormatter on MacÂ OSÂ XÂ 10.4 for more details.

NSDate models the change from the Julian to the Gregorian calendar in October 1582, and calendrical calculations performed in conjunction with NSCalendar take this transition into account. Note, however, that some locales adopted the Gregorian calendar at other times; for example, Great Britain didn't switch over until September 1752.

NSDate is “toll-free bridged” with its Cocoa Foundation counterpart, *CFDate Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an NSDate * parameter, you can pass a CFDateRef, and in a function where you see a CFDateRef parameter, you can pass an NSDate instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Subclassing Notes

The major reason for subclassing NSDate is to create a class with convenience methods for working with a particular calendrical system. But you could also require a custom NSDate class for other reasons, such as to get a date and time value that provides a finer temporal granularity.

Methods to Override

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

- Declare a suitable instance variable to hold the date and time value (relative to an absolute reference date).
- Override the timeIntervalSinceReferenceDate (page 223) instance method to provide the correct date and time value based on your instance variable.
- Override initWithTimeIntervalSinceReferenceDate: (page 220), the designated initializer method.

If you are creating a subclass that represents a calendrical system, you must also define methods that partition past and future periods into the units of this calendar.

Because the NSDate class adopts the NSCopying and NSCoding protocols, your subclass must also implement all of the methods in these protocols.

Special Considerations

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 (page 223) and initWithTimeIntervalSinceReferenceDate: (page 220). 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

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

Tasks

Creating and Initializing Date Objects

+ date (page 213)
  Creates and returns a new date set to the current date and time.

+ dateWithTimeIntervalSinceNow: (page 214)
  Creates and returns an NSDate object set to a given number of seconds from the current date and time.

+ dateWithTimeIntervalSinceReferenceDate: (page 214)
  Creates and returns an NSDate object set to a given number of seconds from the first instant of 1 January 2001, GMT.

+ dateWithTimeIntervalSince1970: (page 213)
  Creates and returns an NSDate object set to the given number of seconds from the first instant of 1 January 1970, GMT.

- init (page 218)
  Returns an NSDate object initialized to the current date and time.

- initWithTimeIntervalSinceNow: (page 219)
  Returns an NSDate object initialized relative to the current date and time by a given number of seconds.

- initWithTimeInterval:sinceDate: (page 219)
  Returns an NSDate object initialized relative to another given date by a given number of seconds.
- initWithTimeIntervalSinceReferenceDate: (page 220)
  Returns an NSDate object initialized relative the first instant of 1 January 2001, GMT by a given number of seconds.

Getting Temporal Boundaries

+ distantFuture (page 215)
  Creates and returns an NSDate object representing a date in the distant future.
+ distantPast (page 215)
  Creates and returns an NSDate object representing a date in the distant past.

Comparing Dates

- isEqualToDate: (page 220)
  Returns a Boolean value that indicates whether a given object is an NSDate object and exactly equal the receiver.
- earlierDate: (page 218)
  Returns the earlier of the receiver and another given date.
- laterDate: (page 221)
  Returns the later of the receiver and another given date.
- compare: (page 217)
  Returns an NSComparisonResult value that indicates the temporal ordering of the receiver and another given date.

Getting Time Intervals

- timeIntervalSinceDate: (page 222)
  Returns the interval between the receiver and another given date.
- timeIntervalSinceNow (page 222)
  Returns the interval between the receiver and the current date and time.
+ timeIntervalSinceReferenceDate (page 215)
  Returns the interval between the first instant of 1 January 2001, GMT and the current date and time.
- timeIntervalSinceReferenceDate (page 223)
  Returns the interval between the receiver and the first instant of 1 January 2001, GMT.
- timeIntervalSince1970 (page 221)
  Returns the interval between the receiver and the first instant of 1 January 1970, GMT.

Adding a Time Interval

- addTimeInterval: (page 216)
  Returns a new NSDate object that is set to a given number of seconds relative to the receiver.

Representing Dates as Strings

- description (page 217)
  Returns a string representation of the receiver in the international format (YYYY-MM-DD HH:MM:SS ±HHMM).

Class Methods

date

Creates and returns a new date set to the current date and time.

```
+ (id)date
```

Return Value
A new date object set to the current date and time.

Discussion
This method uses the default initializer method for the class, init (page 218).

The following code sample shows how to use date to get the current date:

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

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDate.h

dateWithTimeIntervalSince1970:

Creates and returns an NSDate object set to the given number of seconds from the first instant of 1 January 1970, GMT.

```
+ (id)dateWithTimeIntervalSince1970:(NSTimeInterval)seconds
```

Parameters
seconds
  The number of seconds from the reference date, 1 January 1970, GMT, for the new date. Use a negative argument to specify a date before this date.

Return Value
An NSDate object set to *seconds* seconds from the reference date.

Discussion
This method is useful for creating NSDate objects from time_t values returned by BSD system functions.

Availability
Available in iPhone OS 2.0 and later.

See Also
- timeIntervalSince1970 (page 221)

Declared In
NSDate.h

dateWithTimeIntervalSinceNow:

Creates and returns an NSDate object set to a given number of seconds from the current date and time.

```
+ (id)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from the current date and time for the new date. Use a negative value to specify a date before the current date.

Return Value
An NSDate object set to *seconds* seconds from the current date and time.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithTimeIntervalSinceNow: (page 219)

Declared In
NSDate.h

dateWithTimeIntervalSinceReferenceDate:

Creates and returns an NSDate object set to a given number of seconds from the first instant of 1 January 2001, GMT.

```
+ (id)dateWithTimeIntervalSinceReferenceDate:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds from the absolute reference date (the first instant of 1 January 2001, GMT) for the new date. Use a negative argument to specify a date and time before the reference date.

Return Value
An NSDate object set to *seconds* seconds from the absolute reference date.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithTimeIntervalSinceReferenceDate: (page 220)

Declared In
NSDate.h

distantFuture

Creates and returns an `NSDate` object representing a date in the distant future.

```
+ (id)distantFuture
```

Return Value
An `NSDate` object representing a date in the distant future (in terms of centuries).

Discussion
You can pass this value when an `NSDate` object 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];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ distantPast` (page 215)

Declared In
`NSDate.h`

distantPast

Creates and returns an `NSDate` object representing a date in the distant past.

```
+ (id)distantPast
```

Return Value
An `NSDate` object representing a date in the distant past (in terms of centuries).

Discussion
You can use this object as a control date, a guaranteed temporal boundary.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ distantFuture` (page 215)

Declared In
`NSDate.h`

timeIntervalSinceReferenceDate

Returns the interval between the first instant of 1 January 2001, GMT and the current date and time.

```
+ (NSTimeInterval)timeIntervalSinceReferenceDate
```

Return Value
The interval between the system's absolute reference date (the first instant of 1 January 2001, GMT) and the current date and time.

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

Availability
Available in iPhone OS 2.0 and later.

See Also
- `timeIntervalSinceReferenceDate` (page 223)
- `timeIntervalSinceDate:` (page 222)
- `timeIntervalSince1970` (page 221)
- `timeIntervalSinceNow` (page 222)

Declared In
`NSDate.h`

Instance Methods

addTimeInterval:

Returns a new `NSDate` object that is set to a given number of seconds relative to the receiver.

```
- (id)addTimeInterval:(NSTimeInterval)seconds
```

Parameters

seconds
The number of seconds to add to the receiver. Use a negative value for seconds to have the returned object specify a date before the receiver.

Return Value
A new `NSDate` object that is set to *seconds* seconds relative to the receiver. The date returned might have a representation different from the receiver's.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithTimeInterval:sinceDate:` (page 219)
- `timeIntervalSinceDate:` (page 222)

Declared In
`NSDate.h`

compare:

Returns an `NSComparisonResult` value that indicates the temporal ordering of the receiver and another given date.

```
- (NSComparisonResult)compare:(NSDate *)anotherDate
```

Parameters

anotherDate

The date with which to compare the receiver.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

If:

- The receiver and *anotherDate* are exactly equal to each other, `NSOrderedSame`
- The receiver is later in time than *anotherDate*, `NSOrderedDescending`
- The receiver is earlier in time than *anotherDate*, `NSOrderedAscending`.

Discussion

This method detects sub-second differences between dates. If you want to compare dates with a less fine granularity, use `timeIntervalSinceDate:` (page 222) to compare the two dates.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `earlierDate:` (page 218)
- `isEqual:` (page 1306) (`NSObject` protocol)
- `laterDate:` (page 221)

Declared In

`NSDate.h`

description

Returns a string representation of the receiver in the international format (`YYYY-MM-DD HH:MM:SS ±HHMM`).

```
- (NSString *)description
```

Return Value

A string representation of the receiver in 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, “`2001-03-24 10:45:32 +0600`”).

Special Considerations

Important: The `NSDate` implementation of the `description` (page 217) method uses `NSCalendarDate`, which is slated for deprecation. Moreover, `NSCalendarDate` does not model the transition from the Julian to the Gregorian calendar, so the description (created using `NSDate`'s `description` (page 217) method) of a date prior to October 1582 does not match the actual date the object represents. For accurate descriptions, use an instance of `NSDateFormatter` (see *Data Formatting Programming Guide for Cocoa*).

Availability
Available in iPhone OS 2.0 and later.

See Also
`description` (`NSCalendarDate`)

Declared In
`NSDate.h`

earlierDate:

Returns the earlier of the receiver and another given date.

```
- (NSDate *)earlierDate:(NSDate *)anotherDate
```

Parameters
anotherDate
The date with which to compare the receiver.

Return Value
The earlier of the receiver and *anotherDate*, determined using `timeIntervalSinceDate:` (page 222). If the receiver and *anotherDate* represent the same date, returns the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `compare:` (page 217)
- `isEqual:` (page 1306) (`NSObject` protocol)
- `laterDate:` (page 221)

Declared In
`NSDate.h`

init

Returns an `NSDate` object initialized to the current date and time.

```
- (id)init
```

Return Value
An `NSDate` object initialized to the current date and time.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 220).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ date` (page 213)
`- initWithTimeIntervalSinceReferenceDate:` (page 220)

Declared In
`NSDate.h`

initWithTimeInterval:sinceDate:

Returns an `NSDate` object initialized relative to another given date by a given number of seconds.

```
- (id)initWithTimeInterval:(NSTimeInterval)seconds sinceDate:(NSDate *)refDate
```

Parameters

seconds
The number of seconds to add to *refDate*. A negative value means the receiver will be earlier than *refDate*.

refDate
The reference date.

Return Value
An `NSDate` object initialized relative to *refDate* by *seconds* seconds.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 220).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDate.h`

initWithTimeIntervalSinceNow:

Returns an `NSDate` object initialized relative to the current date and time by a given number of seconds.

```
- (id)initWithTimeIntervalSinceNow:(NSTimeInterval)seconds
```

Parameters

seconds
The number of seconds from relative to the current date and time to which the receiver should be initialized. A negative value means the returned object will represent a date in the past.

Return Value
An `NSDate` object initialized relative to the current date and time by *seconds* seconds.

Discussion
This method uses the designated initializer, `initWithTimeIntervalSinceReferenceDate:` (page 220).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dateWithTimeIntervalSinceNow:` (page 214)

Declared In
`NSDate.h`

initWithTimeIntervalSinceReferenceDate:

Returns an `NSDate` object initialized relative the first instant of 1 January 2001, GMT by a given number of seconds.

```
- (id)initWithTimeIntervalSinceReferenceDate:(NSTimeInterval)seconds
```

Parameters

seconds

The number of seconds to add to the reference date (the first instant of 1 January 2001, GMT). A negative value means the receiver will be earlier than the reference date.

Return Value
An `NSDate` object initialized relative to the absolute reference date by *seconds* seconds.

Discussion
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.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dateWithTimeIntervalSinceReferenceDate:` (page 214)

Declared In
`NSDate.h`

isEqualToDate:

Returns a Boolean value that indicates whether a given object is an `NSDate` object and exactly equal the receiver.

```
- (BOOL)isEqualToDate:(NSDate *)anotherDate
```

Parameters

anotherDate

The date to compare with the receiver.

Return Value
YES if the *anotherDate* is an NSDate object and is exactly equal to the receiver, otherwise NO.

Discussion
This method detects sub-second differences between dates. If you want to compare dates with a less fine granularity, use timeIntervalSinceDate: (page 222) to compare the two dates.

Availability
Available in iPhone OS 2.0 and later.

See Also
- compare: (page 217)
- earlierDate: (page 218)
- isEqual: (page 1306) (NSObject protocol)
- laterDate: (page 221)

Declared In
NSDate.h

laterDate:
Returns the later of the receiver and another given date.

```
- (NSDate *)laterDate:(NSDate *)anotherDate
```

Parameters
anotherDate
The date with which to compare the receiver.

Return Value
The later of the receiver and *anotherDate*, determined using timeIntervalSinceDate: (page 222). If the receiver and *anotherDate* represent the same date, returns the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- compare: (page 217)
- earlierDate: (page 218)
- isEqual: (page 1306) (NSObject protocol)

Declared In
NSDate.h

timeIntervalSince1970
Returns the interval between the receiver and the first instant of 1 January 1970, GMT.

```
- (NSTimeInterval)timeIntervalSince1970
```

Return Value
The interval between the receiver and the reference date, 1 January 1970, GMT. If the receiver is earlier than the reference date, the value is negative.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- timeIntervalSinceDate:` (page 222)
- `- timeIntervalSinceNow` (page 222)
- `- timeIntervalSinceReferenceDate` (page 223)
- `+ timeIntervalSinceReferenceDate` (page 215)

Declared In
`NSDate.h`

timeIntervalSinceDate:

Returns the interval between the receiver and another given date.

```
- (NSTimeInterval)timeIntervalSinceDate:(NSDate *)anotherDate
```

Parameters
anotherDate
The date with which to compare the receiver.

Return Value
The interval between the receiver and *anotherDate*. If the receiver is earlier than *anotherDate*, the return value is negative.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- timeIntervalSince1970` (page 221)
- `- timeIntervalSinceNow` (page 222)
- `- timeIntervalSinceReferenceDate` (page 223)

Declared In
`NSDate.h`

timeIntervalSinceNow

Returns the interval between the receiver and the current date and time.

```
- (NSTimeInterval)timeIntervalSinceNow
```

Return Value
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.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- timeIntervalSinceDate:` (page 222)
- `- timeIntervalSince1970` (page 221)

- timeIntervalSinceReferenceDate (page 223)

Declared In
NSDate.h

timeIntervalSinceReferenceDate

Returns the interval between the receiver and the first instant of 1 January 2001, GMT.

```
- (NSTimeInterval)timeIntervalSinceReferenceDate
```

Return Value
The interval between the receiver and the system's absolute reference date (the first instant of 1 January 2001, GMT). If the receiver is earlier than the reference date, the value is negative.

Availability
Available in iPhone OS 2.0 and later.

See Also
- timeIntervalSinceDate: (page 222)
- timeIntervalSinceNow (page 222)
+ timeIntervalSinceReferenceDate (page 215)

Declared In
NSDate.h

Constants

NSTimeIntervalSince1970

NSDate provides a constant that specifies the number of seconds from 1 January 1970 to the reference date, 1 January 2001.

```
#define NSTimeIntervalSince1970  978307200.0
```

Constants
NSTimeIntervalSince1970
The number of seconds from 1 January 1970 to the reference date, 1 January 2001.
Available in iPhone OS 2.0 and later.
Declared in NSDate.h

Discussion
1 January 1970 is the epoch (or starting point) for Unix time.

Declared In
NSDate.h

NSDateComponents Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSCalendar.h
Companion guide:	Date and Time Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSDateComponents` encapsulates the components of a date in an extendable, object-oriented manner. It is used to specify a date by providing the temporal components that make up a date and time: hour, minutes, seconds, day, month, year, and so on. It can also be used to specify a duration of time, for example, 5 hours and 16 minutes. An `NSDateComponents` object is not required to define all the component fields. When a new instance of `NSDateComponents` is created the date components are set to `NSUndefinedDateComponent`.

Important: An `NSDateComponents` object 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.

An instance of `NSDateComponents` is not responsible for answering questions about a date beyond the information with which it was initialized. For example, if you initialize one with May 6, 2004, its weekday is `NSUndefinedDateComponent`, not Thursday. To get the correct day of the week, you must create a suitable instance of `NSCalendar`, create an `NSDate` object using `dateFromComponents:` and then use `components:fromDate:` to retrieve the weekday—as illustrated in the following example.

```
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];
NSDateComponents *weekdayComponents =
    [gregorian components:NSWeekdayCalendarUnit fromDate:date];
int weekday = [weekdayComponents weekday];
```

For more details, see Calendars in *Date and Time Programming Guide for Cocoa*.

Tasks

Getting Information About an NSDateComponents Object

- `era` (page 228)
  Returns the number of era units for the receiver.
- `year` (page 235)
  Returns the number of year units for the receiver.
- `month` (page 229)
  Returns the number of month units for the receiver.
- `day` (page 227)
  Returns the number of day units for the receiver.
- `hour` (page 228)
  Returns the number of hour units for the receiver.
- `minute` (page 228)
  Returns the number of minute units for the receiver.
- `second` (page 229)
  Returns the number of second units for the receiver.
- `week` (page 234)
  Returns the number of week units for the receiver.
- `weekday` (page 234)
  Returns the number of weekday units for the receiver.

- `weekdayOrdinal` (page 235)
  Returns the ordinal number of weekday units for the receiver.

Setting Information for an NSDateComponents Object

- `setEra:` (page 230)
  Sets the number of era units for the receiver.
- `setYear:` (page 234)
  Sets the number of year units for the receiver.
- `setMonth:` (page 231)
  Sets the number of month units for the receiver.
- `setDay:` (page 230)
  Sets the number of day units for the receiver.
- `setHour:` (page 231)
  Sets the number of hour units for the receiver.
- `setMinute:` (page 231)
  Sets the number of minute units for the receiver.
- `setSecond:` (page 232)
  Sets the number of second units for the receiver.
- `setWeek:` (page 232)
  Sets the number of week units for the receiver.
- `setWeekday:` (page 233)
  Sets the number of weekday units for the receiver.
- `setWeekdayOrdinal:` (page 233)
  Sets the ordinal number of weekday units for the receiver.

Instance Methods

day

Returns the number of day units for the receiver.

```
- (NSInteger)day
```

Return Value
The number of day units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDay:` (page 230)

Declared In
NSCalendar.h

era

Returns the number of era units for the receiver.

```
- (NSInteger)era
```

Return Value
The number of era units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setEra: (page 230)

Declared In
NSCalendar.h

hour

Returns the number of hour units for the receiver.

```
- (NSInteger)hour
```

Return Value
The number of hour units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setHour: (page 231)

Declared In
NSCalendar.h

minute

Returns the number of minute units for the receiver.

```
- (NSInteger)minute
```

Return Value
The number of minute units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setMinute:` (page 231)

Declared In
`NSCalendar.h`

month

Returns the number of month units for the receiver.

```
- (NSInteger)month
```

Return Value
The number of month units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setMonth:` (page 231)

Declared In
`NSCalendar.h`

second

Returns the number of second units for the receiver.

```
- (NSInteger)second
```

Return Value
The number of second units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setSecond: (page 232)

Declared In
NSCalendar.h

setDay:

Sets the number of day units for the receiver.

```
- (void)setDay:(NSInteger)v
```

Parameters

v
The number of day units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- day (page 227)

Declared In
NSCalendar.h

setEra:

Sets the number of era units for the receiver.

```
- (void)setEra:(NSInteger)v
```

Parameters

v
The number of era units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- era (page 228)

Declared In
NSCalendar.h

setHour:

Sets the number of hour units for the receiver.

```
- (void)setHour:(NSInteger)v
```

Parameters

v

The number of hour units for the receiver.

Discussion

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- hour (page 228)

Declared In

NSCalendar.h

setMinute:

Sets the number of minute units for the receiver.

```
- (void)setMinute:(NSInteger)v
```

Parameters

v

The number of minute units for the receiver.

Discussion

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- minute (page 228)

Declared In

NSCalendar.h

setMonth:

Sets the number of month units for the receiver.

```
- (void)setMonth:(NSInteger)v
```

Parameters

v

The number of month units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- month (page 229)

Declared In
NSCalendar.h

setSecond:

Sets the number of second units for the receiver.

```
- (void)setSecond:(NSInteger)v
```

Parameters

v
The number of second units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- second (page 229)

Declared In
NSCalendar.h

setWeek:

Sets the number of week units for the receiver.

```
- (void)setWeek:(NSInteger)v
```

Parameters

v
The number of week units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- week (page 234)

Declared In
NSCalendar.h

setWeekday:

Sets the number of weekday units for the receiver.

```
- (void)setWeekday:(NSInteger)v
```

Parameters

v
The number of weekday units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- weekday (page 234)

Declared In
NSCalendar.h

setWeekdayOrdinal:

Sets the ordinal number of weekday units for the receiver.

```
- (void)setWeekdayOrdinal:(NSInteger)v
```

Parameters

v
The ordinal number of weekday units for the receiver.

Discussion
This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- weekdayOrdinal (page 235)

Declared In
NSCalendar.h

setYear:

Sets the number of year units for the receiver.

```
- (void)setYear:(NSInteger)v
```

Parameters

v

The number of year units for the receiver.

Discussion

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- year (page 235)

Declared In

NSCalendar.h

week

Returns the number of week units for the receiver.

```
- (NSInteger)week
```

Return Value

The number of week units for the receiver.

Discussion

This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setWeek: (page 232)

Declared In

NSCalendar.h

weekday

Returns the number of weekday units for the receiver.

```
- (NSInteger)weekday
```

Return Value

The number of weekday units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setWeekday:` (page 233)

Declared In
`NSCalendar.h`

weekdayOrdinal

Returns the ordinal number of weekday units for the receiver.

```
- (NSInteger)weekdayOrdinal
```

Return Value
The ordinal number of weekday units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setWeekdayOrdinal:` (page 233)

Declared In
`NSCalendar.h`

year

Returns the number of year units for the receiver.

```
- (NSInteger)year
```

Return Value
The number of year units for the receiver.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setYear:` (page 234)

Declared In
NSCalendar.h

Constants

NSDateComponents undefined component identifier

This constant specifies that an NSDateComponents component is undefined.

```
enum {
    NSUndefinedDateComponent = 0x7fffffff
};
```

Constants
NSUndefinedDateComponent
Specifies that the component is undefined.
Available in iPhone OS 2.0 and later.
Declared in NSCalendar.h

Declared In
NSCalendar.h

NSDateFormatter Class Reference

Inherits from:	NSFormatter : NSObject
Conforms to:	NSObject (NSObject) NSCoding (NSFormatter) NSCopying (NSFormatter)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDateFormatter.h
Companion guide:	Data Formatting Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

Instances of `NSDateFormatter` create string representations of `NSDate` (and `NSCalendarDate`) objects, and convert textual representations of dates and times into `NSDate` objects. You can express the representation of dates and times flexibly: “Thu 22 Dec 1994” is just as acceptable as “12/22/94.”

With Mac OS X v10.4 and later, `NSDateFormatter` has two modes of operation (or behaviors). By default, instances of `NSDateFormatter` have the same behavior as they did on Mac OS X versions 10.0 to 10.3. You can, however, configure instances (or set a default for all instances) to adopt a new behavior implemented for Mac OS X version 10.4. See *Data Formatting Programming Guide for Cocoa* for a full description of the old and new behaviors.

If you initialize a formatter using `initWithDateFormat:allowNaturalLanguage:`, you are (for backwards compatibility reasons) creating an "old-style" date formatter. To use the new behavior, you initialize the formatter with `init` (page 248). If you have not set the default class behavior (see `setDefaultFormatterBehavior:` (page 243)), you send the instance a `setFormatterBehavior:` (page 253) message with the argument `NSDateFormatterBehavior10_4`. You can then set the date format as appropriate, typically using a format style as illustrated in the following code fragment.

```
// assume default behavior set for class using
// [NSDateFormatter setDefaultFormatterBehavior:NSDateFormatterBehavior10_4];

NSDateFormatter *dateFormatter = [[[NSDateFormatter alloc] init]  autorelease];
[dateFormatter setDateStyle:NSDateFormatterMediumStyle];
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];

NSDate *date = [NSDate dateWithTimeIntervalSinceReferenceDate:118800];
NSString *formattedDateString = [dateFormatter stringFromDate:date];
NSLog(@"formattedDateString for locale %@: %@",
        [[dateFormatter locale] localeIdentifier], formattedDateString);

// Output: formattedDateString for locale en_US: Jan 2, 2001
```

Note that the format for a given style is dependent on a user's preferences, including the locale setting.

Note also that by default the new-style formatter returns `NSDate` objects instead of `NSCalendarDate` objects. You can change this behavior using `setGeneratesCalendarDates:` (page 253).

Important: Date formatters created in Interface Builder are initialized with `initWithDateFormat:allowNaturalLanguage:`, and hence use the Mac OS X version 10.0 behavior.

Tasks

Initializing a Date Formatter

- `init` (page 248)
  Initializes and returns an `NSDateFormatter` instance.

Managing Behavior

- `formatterBehavior` (page 246)
  Returns the formatter behavior for the receiver.
- `setFormatterBehavior:` (page 253)
  Sets the formatter behavior for the receiver.
+ `defaultFormatterBehavior` (page 242)
  Returns the default formatting behavior for instances of the class.
+ `setDefaultFormatterBehavior:` (page 243)
  Sets the default formatting behavior for instances of the class.

- `generatesCalendarDates` (page 246)
  Returns a Boolean value that indicates whether the receiver generates calendar dates.
- `setGeneratesCalendarDates:` (page 253)
  Sets whether the receiver generates calendar dates.
- `isLenient` (page 248)
  Returns a Boolean value that indicates whether the receiver uses heuristics when parsing a string.
- `setLenient:` (page 254)
  Sets whether the receiver uses heuristics when parsing a string.

Converting Objects

- `dateFromString:` (page 244)
  Returns a date representation of a given string interpreted using the receiver's current settings.
- `stringFromDate:` (page 268)
  Returns a string representation of a given date formatted using the receiver's current settings.
- `getObjectValue:forString:range:error:` (page 247)
  Returns by reference a date representation of a given string and the range of the string used, and returns a Boolean value that indicates whether the string could be parsed.

Managing Formats and Styles

- `dateFormat` (page 244)
  Returns the date format string used by the receiver.
- `setDateFormat:` (page 251)
  Sets the date format for the receiver.
- `dateStyle` (page 245)
  Returns the date style of the receiver.
- `setDateStyle:` (page 252)
  Sets the date style of the receiver.
- `timeStyle` (page 268)
  Returns the time style of the receiver.
- `setTimeStyle:` (page 261)
  Sets the time style of the receiver.

Managing Attributes

- `calendar` (page 244)
  Returns the calendar for the receiver.
- `setCalendar:` (page 251)
  Sets the calendar for the receiver.
- `defaultDate` (page 245)
  Returns the default date for the receiver.

- `setDefaultDate:` (page 252)
  Sets the default date for the receiver.
- `locale` (page 249)
  Returns the locale for the receiver.
- `setLocale:` (page 254)
  Sets the locale for the receiver.
- `timeZone` (page 269)
  Returns the time zone for the receiver.
- `setTimeZone:` (page 261)
  Sets the time zone for the receiver.
- `twoDigitStartDate` (page 269)
  Returns the earliest date that can be denoted by a two-digit year specifier.
- `setTwoDigitStartDate:` (page 261)
  Sets the two-digit start date for the receiver.
- `gregorianStartDate` (page 247)
  Returns the start date of the Gregorian calendar for the receiver.
- `setGregorianStartDate:` (page 254)
  Sets the start date of the Gregorian calendar for the receiver.

Managing AM and PM Symbols

- `AMSymbol` (page 243)
  Returns the AM symbol for the receiver.
- `setAMSymbol:` (page 250)
  Sets the AM symbol for the receiver.
- `PMSymbol` (page 250)
  Returns the PM symbol for the receiver.
- `setPMSymbol:` (page 256)
  Sets the PM symbol for the receiver.

Managing Weekday Symbols

- `weekdaySymbols` (page 271)
  Returns the array of weekday symbols for the receiver.
- `setWeekdaySymbols:` (page 263)
  Sets the weekday symbols for the receiver.
- `shortWeekdaySymbols` (page 266)
  Returns the array of short weekday symbols for the receiver.
- `setShortWeekdaySymbols:` (page 259)
  Sets the short weekday symbols for the receiver.
- `veryShortWeekdaySymbols` (page 271)
  Returns the array of very short weekday symbols for the receiver.
- `setVeryShortWeekdaySymbols:` (page 263)
  Sets the vert short weekday symbols for the receiver

- standaloneWeekdaySymbols (page 267)
  Returns the array of standalone weekday symbols for the receiver.
- setStandaloneWeekdaySymbols: (page 260)
  Sets the standalone weekday symbols for the receiver.
- shortStandaloneWeekdaySymbols (page 266)
  Returns the array of short standalone weekday symbols for the receiver.
- setShortStandaloneWeekdaySymbols: (page 258)
  Sets the short standalone weekday symbols for the receiver.
- veryShortStandaloneWeekdaySymbols (page 270)
  Returns the array of very short standalone weekday symbols for the receiver.
- setVeryShortStandaloneWeekdaySymbols: (page 263)
  Sets the very short standalone weekday symbols for the receiver.

Managing Month Symbols

- monthSymbols (page 249)
  Returns the month symbols for the receiver.
- setMonthSymbols: (page 255)
  Sets the month symbols for the receiver.
- shortMonthSymbols (page 264)
  Returns the array of short month symbols for the receiver.
- setShortMonthSymbols: (page 256)
  Sets the short month symbols for the receiver.
- veryShortMonthSymbols (page 269)
  Returns the very short month symbols for the receiver.
- setVeryShortMonthSymbols: (page 262)
  Sets the very short month symbols for the receiver.
- standaloneMonthSymbols (page 267)
  Returns the standalone month symbols for the receiver.
- setStandaloneMonthSymbols: (page 259)
  Sets the standalone month symbols for the receiver.
- shortStandaloneMonthSymbols (page 265)
  Returns the short standalone month symbols for the receiver.
- setShortStandaloneMonthSymbols: (page 257)
  Sets the short standalone month symbols for the receiver.
- veryShortStandaloneMonthSymbols (page 270)
  Returns the very short month symbols for the receiver.
- setVeryShortStandaloneMonthSymbols: (page 262)
  Sets the very short standalone month symbols for the receiver.

Managing Quarter Symbols

- quarterSymbols (page 250)
  Returns the quarter symbols for the receiver.

- `setQuarterSymbols:` (page 256)
  Sets the quarter symbols for the receiver.
- `shortQuarterSymbols` (page 264)
  Returns the short quarter symbols for the receiver.
- `setShortQuarterSymbols:` (page 257)
  Sets the short quarter symbols for the receiver.
- `standaloneQuarterSymbols` (page 267)
  Returns the standalone quarter symbols for the receiver.
- `setStandaloneQuarterSymbols:` (page 260)
  Sets the standalone quarter symbols for the receiver.
- `shortStandaloneQuarterSymbols` (page 265)
  Returns the short standalone quarter symbols for the receiver.
- `setShortStandaloneQuarterSymbols:` (page 258)
  Sets the short standalone quarter symbols for the receiver.

Managing Era Symbols

- `eraSymbols` (page 245)
  Returns the era symbols for the receiver.
- `setEraSymbols:` (page 252)
  Sets the era symbols for the receiver.
- `longEraSymbols` (page 249)
  Returns the long era symbols for the receiver
- `setLongEraSymbols:` (page 255)
  Sets the long era symbols for the receiver.

Class Methods

defaultFormatterBehavior

Returns the default formatting behavior for instances of the class.

```
+ (NSDateFormatterBehavior)defaultFormatterBehavior
```

Return Value

The default formatting behavior for instances of the class. For possible values, see NSDateFormatterBehavior (page 273).

Discussion

The default is `NSDateFormatterBehavior10_0`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ setDefaultFormatterBehavior:` (page 243).

- formatterBehavior (page 246)
- setFormatterBehavior: (page 253)

Declared In
NSDateFormatter.h

setDefaultFormatterBehavior:

Sets the default formatting behavior for instances of the class.

```
+ (void)setDefaultFormatterBehavior:(NSDateFormatterBehavior)behavior
```

Parameters

behavior

The default formatting behavior for instances of the class. For possible values, see NSDateFormatterBehavior (page 273).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ defaultFormatterBehavior (page 242)
- formatterBehavior (page 246)
- setFormatterBehavior: (page 253)

Declared In
NSDateFormatter.h

Instance Methods

AMSymbol

Returns the AM symbol for the receiver.

```
- (NSString *)AMSymbol
```

Return Value
The AM symbol for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setAMSymbol: (page 250)
- PMSymbol (page 250)
- setPMSymbol: (page 256)

Declared In
NSDateFormatter.h

calendar

Returns the calendar for the receiver.

```
- (NSCalendar *)calendar
```

Return Value
The calendar for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setCalendar: (page 251)

Declared In
NSDateFormatter.h

dateFormat

Returns the date format string used by the receiver.

```
- (NSString *)dateFormat
```

Return Value
The date format string used by the receiver.

Discussion
See Date Format String Syntax (Mac OS X Versions 10.0 to 10.3) for a list of the conversion specifiers permitted in date format strings.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setDateFormat: (page 251)

Declared In
NSDateFormatter.h

dateFromString:

Returns a date representation of a given string interpreted using the receiver's current settings.

```
- (NSDate *)dateFromString:(NSString *)string
```

Parameters
string
The string to parse.

Return Value
A date representation of *string* interpreted using the receiver's current settings.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `getObjectValue:forString:range:error:` (page 247)
- `stringFromDate:` (page 268)

Declared In
`NSDateFormatter.h`

dateStyle

Returns the date style of the receiver.

```
- (NSDateFormatterStyle)dateStyle
```

Return Value
The date style of the receiver. For possible values, see NSDateFormatterStyle (page 272).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDateStyle:` (page 252)

Declared In
`NSDateFormatter.h`

defaultDate

Returns the default date for the receiver.

```
- (NSDate *)defaultDate
```

Return Value
The default date for the receiver.

Discussion
The default default date is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDefaultDate:` (page 252)

Declared In
`NSDateFormatter.h`

eraSymbols

Returns the era symbols for the receiver.

```
- (NSArray *)eraSymbols
```

Return Value
An array containing `NSString` objects representing the era symbols for the receiver (for example, {“B.C.E.”, “C.E.”}).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setEraSymbols:` (page 252)
- `longEraSymbols` (page 249)

Declared In
`NSDateFormatter.h`

formatterBehavior
Returns the formatter behavior for the receiver.

```
- (NSDateFormatterBehavior)formatterBehavior
```

Return Value
The formatter behavior for the receiver. For possible values, see `NSDateFormatterBehavior` (page 273).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ `defaultFormatterBehavior` (page 242).
+ `setDefaultFormatterBehavior:` (page 243)
- `setFormatterBehavior:` (page 253)

Declared In
`NSDateFormatter.h`

generatesCalendarDates
Returns a Boolean value that indicates whether the receiver generates calendar dates.

```
- (BOOL)generatesCalendarDates
```

Return Value
`YES` if the receiver generates calendar dates, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setGeneratesCalendarDates:` (page 253)

Declared In
`NSDateFormatter.h`

getObjectValue:forString:range:error:

Returns by reference a date representation of a given string and the range of the string used, and returns a Boolean value that indicates whether the string could be parsed.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string range:(inout NSRange
    *)rangep error:(NSError **)error
```

Parameters

obj

If the receiver is able to parse *string*, upon return contains a date representation of *string*.

string

The string to parse.

rangep

If the receiver is able to parse *string*, upon return contains the range of *string* used to create the date.

error

If the receiver is unable to create a date by parsing *string*, upon return contains an NSError object that describes the problem.

Return Value

`YES` if the receiver can create a date by parsing *string*, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `dateFromString:` (page 244)
- `stringForObjectValue:` (page 433)

Declared In

`NSDateFormatter.h`

gregorianStartDate

Returns the start date of the Gregorian calendar for the receiver.

```
- (NSDate *)gregorianStartDate
```

Return Value

The start date of the Gregorian calendar for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setGregorianStartDate:` (page 254)

Declared In

`NSDateFormatter.h`

init

Initializes and returns an `NSDateFormatter` instance.

```
- (id)init
```

Return Value

An `NSDateFormatter` instance initialized with locale, time zone, calendar, and behavior set to the appropriate default values.

Discussion

There are many new attributes you can get and set on a 10.4-style date formatter, including the locale, time zone, calendar, format string, the two-digit-year cross-over date, the default date which provides unspecified components, and there is also access to the various textual strings, like the month names. You are encouraged, however, not to change individual settings. Instead you should accept the default settings established on initialization and specify the format using `setDateStyle:` (page 252), `setTimeStyle:` (page 261), and appropriate style constants (see `NSDateFormatterStyle` (page 272)—these are styles that the user can configure in the International preferences panel in System Preferences).

Special Considerations

If you want the Mac OS X 10.4 behavior but have not set the class's default behavior to `NSDateFormatterBehavior10_4`, you also need to send the new instance a `setFormatterBehavior:` (page 253) message with the argument `NSDateFormatterBehavior10_4`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setDateStyle:` (page 252)
- `setTimeStyle:` (page 261)

Declared In

`NSDateFormatter.h`

isLenient

Returns a Boolean value that indicates whether the receiver uses heuristics when parsing a string.

```
- (BOOL)isLenient
```

Return Value

`YES` if the receiver has been set to use heuristics when parsing a string to guess at the date which is intended, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setLenient:` (page 254)

Declared In

`NSDateFormatter.h`

locale

Returns the locale for the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setLocale:` (page 254)

Declared In
`NSDateFormatter.h`

longEraSymbols

Returns the long era symbols for the receiver

```
- (NSArray *)longEraSymbols
```

Return Value
An array containing `NSString` objects representing the era symbols for the receiver (for example, {“Before Common Era”, “Common Era”}).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setLongEraSymbols:` (page 255)
- `eraSymbols` (page 245)

Declared In
`NSDateFormatter.h`

monthSymbols

Returns the month symbols for the receiver.

```
- (NSArray *)monthSymbols
```

Return Value
An array of `NSString` objects that specify the month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMonthSymbols:` (page 255)
- `shortMonthSymbols` (page 264)
- `veryShortMonthSymbols` (page 269)

- standaloneMonthSymbols (page 267)
- shortStandaloneMonthSymbols (page 265)
- veryShortStandaloneMonthSymbols (page 270)

Declared In
NSDateFormatter.h

PMSymbol

Returns the PM symbol for the receiver.

```
- (NSString *)PMSymbol
```

Return Value
The PM symbol for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPMSymbol: (page 256)
- AMSymbol (page 243)
- setAMSymbol: (page 250)

Declared In
NSDateFormatter.h

quarterSymbols

Returns the quarter symbols for the receiver.

```
- (NSArray *)quarterSymbols
```

Return Value
An array containing NSString objects representing the quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setQuarterSymbols: (page 256)
- shortQuarterSymbols (page 264)
- standaloneQuarterSymbols (page 267)
- shortStandaloneQuarterSymbols (page 265)

Declared In
NSDateFormatter.h

setAMSymbol:

Sets the AM symbol for the receiver.

```
- (void)setAMSymbol:(NSString *)string
```

Parameters

string

The AM symbol for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- AMSymbol (page 243)
- PMSymbol (page 250)
- setPMSymbol: (page 256)

Declared In

NSDateFormatter.h

setCalendar:

Sets the calendar for the receiver.

```
- (void)setCalendar:(NSCalendar *)calendar
```

Parameters

calendar

The calendar for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- calendar (page 244)

Declared In

NSDateFormatter.h

setDateFormat:

Sets the date format for the receiver.

```
- (void)setDateFormat:(NSString *)string
```

Parameters

string

The date format for the receiver. See *Data Formatting Programming Guide for Cocoa* for a list of the conversion specifiers permitted in date format strings.

Availability

Available in iPhone OS 2.0 and later.

See Also

- dateFormat (page 244).

Declared In
NSDateFormatter.h

setDateStyle:

Sets the date style of the receiver.

```
- (void)setDateStyle:(NSDateFormatterStyle)style
```

Parameters

style

The date style of the receiver. For possible values, see NSDateFormatterStyle (page 272).

Availability
Available in iPhone OS 2.0 and later.

See Also
- dateStyle (page 245).

Declared In
NSDateFormatter.h

setDefaultDate:

Sets the default date for the receiver.

```
- (void)setDefaultDate:(NSDate *)date
```

Parameters

date

The default date for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- defaultDate (page 245)

Declared In
NSDateFormatter.h

setEraSymbols:

Sets the era symbols for the receiver.

```
- (void)setEraSymbols:(NSArray *)array
```

Parameters

array

An array containing NSString objects representing the era symbols for the receiver (for example, {“B.C.E.”, “C.E.”}).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `eraSymbols` (page 245)
- `longEraSymbols` (page 249)

Declared In
`NSDateFormatter.h`

setFormatterBehavior:

Sets the formatter behavior for the receiver.

```
- (void)setFormatterBehavior:(NSDateFormatterBehavior)behavior
```

Parameters

behavior

The formatter behavior for the receiver. For possible values, see `NSDateFormatterBehavior` (page 273).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `+ defaultFormatterBehavior` (page 242).
- `+ setDefaultFormatterBehavior:` (page 243)
- `- formatterBehavior` (page 246)

Declared In
`NSDateFormatter.h`

setGeneratesCalendarDates:

Sets whether the receiver generates calendar dates.

```
- (void)setGeneratesCalendarDates:(BOOL)b
```

Parameters

b

A Boolean value that specifies whether the receiver generates calendar dates.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `generatesCalendarDates` (page 246).

Declared In
`NSDateFormatter.h`

setGregorianStartDate:

Sets the start date of the Gregorian calendar for the receiver.

```
- (void)setGregorianStartDate:(NSDate *)array
```

Parameters

array

The start date of the Gregorian calendar for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- gregorianStartDate (page 247)

Declared In

NSDateFormatter.h

setLenient:

Sets whether the receiver uses heuristics when parsing a string.

```
- (void)setLenient:(BOOL)b
```

Parameters

b

YES to use heuristics when parsing a string to guess at the date which is intended, otherwise NO.

Discussion

If a formatter is set to be lenient, when parsing a string it uses heuristics to guess at the date which is intended. As with any guessing, it may get the result date wrong (that is, a date other than that which was intended).

Availability

Available in iPhone OS 2.0 and later.

See Also

- isLenient (page 248)

Declared In

NSDateFormatter.h

setLocale:

Sets the locale for the receiver.

```
- (void)setLocale:(NSLocale *)locale
```

Parameters

locale

The locale for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `locale` (page 249)

Declared In
`NSDateFormatter.h`

setLongEraSymbols:

Sets the long era symbols for the receiver.

```
- (void)setLongEraSymbols:(NSArray *)array
```

Parameters

array

An array containing `NSString` objects representing the era symbols for the receiver (for example, {“Before Common Era”, “Common Era”}).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `longEraSymbols` (page 249)
- `eraSymbols` (page 245)

Declared In
`NSDateFormatter.h`

setMonthSymbols:

Sets the month symbols for the receiver.

```
- (void)setMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `monthSymbols` (page 249)
- `setShortMonthSymbols:` (page 256)
- `setVeryShortMonthSymbols:` (page 262)
- `setStandaloneMonthSymbols:` (page 259)
- `setShortStandaloneMonthSymbols:` (page 257)
- `setVeryShortStandaloneMonthSymbols:` (page 262)

Declared In
NSDateFormatter.h

setPMSymbol:

Sets the PM symbol for the receiver.

```
- (void)setPMSymbol:(NSString *)string
```

Parameters

string
The PM symbol for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- PMSymbol (page 250)
- AMSymbol (page 243)
- setAMSymbol: (page 250)

Declared In
NSDateFormatter.h

setQuarterSymbols:

Sets the quarter symbols for the receiver.

```
- (void)setQuarterSymbols:(NSArray *)array
```

Parameters

array
An array of NSString objects that specify the quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- quarterSymbols (page 250)
- setShortQuarterSymbols: (page 257)
- setStandaloneQuarterSymbols: (page 260)
- setShortStandaloneQuarterSymbols: (page 258)

Declared In
NSDateFormatter.h

setShortMonthSymbols:

Sets the short month symbols for the receiver.

```
- (void)setShortMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the short month symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `shortMonthSymbols` (page 264)
- `setMonthSymbols:` (page 255)
- `setVeryShortMonthSymbols:` (page 262)
- `setStandaloneMonthSymbols:` (page 259)
- `setShortStandaloneMonthSymbols:` (page 257)
- `setVeryShortStandaloneMonthSymbols:` (page 262)

Declared In

`NSDateFormatter.h`

setShortQuarterSymbols:

Sets the short quarter symbols for the receiver.

```
- (void)setShortQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the short quarter symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `shortQuarterSymbols` (page 264)
- `setQuarterSymbols:` (page 256)
- `setStandaloneQuarterSymbols:` (page 260)
- `setShortStandaloneQuarterSymbols:` (page 258)

Declared In

`NSDateFormatter.h`

setShortStandaloneMonthSymbols:

Sets the short standalone month symbols for the receiver.

```
- (void)setShortStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the short standalone month symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also
- shortStandaloneMonthSymbols (page 265)
- setMonthSymbols: (page 255)
- setShortMonthSymbols: (page 256)
- setVeryShortMonthSymbols: (page 262)
- setStandaloneMonthSymbols: (page 259)
- setVeryShortStandaloneMonthSymbols: (page 262)

Declared In
NSDateFormatter.h

setShortStandaloneQuarterSymbols:

Sets the short standalone quarter symbols for the receiver.

```
- (void)setShortStandaloneQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the short standalone quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- shortStandaloneQuarterSymbols (page 265)
- setQuarterSymbols: (page 256)
- setShortQuarterSymbols: (page 257)
- setStandaloneQuarterSymbols: (page 260)

Declared In
NSDateFormatter.h

setShortStandaloneWeekdaySymbols:

Sets the short standalone weekday symbols for the receiver.

```
- (void)setShortStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the short standalone weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- shortStandaloneWeekdaySymbols (page 266)
- setWeekdaySymbols: (page 263)
- setShortWeekdaySymbols: (page 259)

- setVeryShortWeekdaySymbols: (page 263)
- setStandaloneWeekdaySymbols: (page 260)
- setVeryShortStandaloneWeekdaySymbols: (page 263)

Declared In
NSDateFormatter.h

setShortWeekdaySymbols:

Sets the short weekday symbols for the receiver.

```
- (void)setShortWeekdaySymbols:(NSArray *)array
```

Parameters

array
An array of NSString objects that specify the short weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- shortWeekdaySymbols (page 266)
- setWeekdaySymbols: (page 263)
- setVeryShortWeekdaySymbols: (page 263)
- setStandaloneWeekdaySymbols: (page 260)
- setShortStandaloneWeekdaySymbols: (page 258)
- setVeryShortStandaloneWeekdaySymbols: (page 263)

Declared In
NSDateFormatter.h

setStandaloneMonthSymbols:

Sets the standalone month symbols for the receiver.

```
- (void)setStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array
An array of NSString objects that specify the standalone month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- standaloneMonthSymbols (page 267)
- setMonthSymbols: (page 255)
- setShortMonthSymbols: (page 256)
- setVeryShortMonthSymbols: (page 262)
- setShortStandaloneMonthSymbols: (page 257)
- setVeryShortStandaloneMonthSymbols: (page 262)

Declared In
NSDateFormatter.h

setStandaloneQuarterSymbols:

Sets the standalone quarter symbols for the receiver.

```
- (void)setStandaloneQuarterSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the standalone quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setStandaloneQuarterSymbols: (page 260)
- setQuarterSymbols: (page 256)
- setShortQuarterSymbols: (page 257)
- setShortStandaloneQuarterSymbols: (page 258)

Declared In
NSDateFormatter.h

setStandaloneWeekdaySymbols:

Sets the standalone weekday symbols for the receiver.

```
- (void)setStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the standalone weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- standaloneWeekdaySymbols (page 267)
- setWeekdaySymbols: (page 263)
- setShortWeekdaySymbols: (page 259)
- setVeryShortWeekdaySymbols: (page 263)
- setShortStandaloneWeekdaySymbols: (page 258)
- setVeryShortStandaloneWeekdaySymbols: (page 263)

Declared In
NSDateFormatter.h

setTimeStyle:

Sets the time style of the receiver.

```
- (void)setTimeStyle:(NSDateFormatterStyle)style
```

Parameters

style

The time style for the receiver. For possible values, see NSDateFormatterStyle (page 272).

Availability

Available in iPhone OS 2.0 and later.

See Also

- timeStyle (page 268)

Declared In

NSDateFormatter.h

setTimeZone:

Sets the time zone for the receiver.

```
- (void)setTimeZone:(NSTimeZone *)tz
```

Parameters

tz

The time zone for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- timeZone (page 269)

Declared In

NSDateFormatter.h

setTwoDigitStartDate:

Sets the two-digit start date for the receiver.

```
- (void)setTwoDigitStartDate:(NSDate *)date
```

Parameters

date

The earliest date that can be denoted by a two-digit year specifier.

Availability

Available in iPhone OS 2.0 and later.

See Also

- twoDigitStartDate (page 269)

Declared In
NSDateFormatter.h

setVeryShortMonthSymbols:

Sets the very short month symbols for the receiver.

```
- (void)setVeryShortMonthSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the very short month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- verySymbolMonthSymbols (page 269)
- setMonthSymbols: (page 255)
- setShortMonthSymbols: (page 256)
- setStandaloneMonthSymbols: (page 259)
- setShortStandaloneMonthSymbols: (page 257)
- setVeryShortStandaloneMonthSymbols: (page 262)

Declared In
NSDateFormatter.h

setVeryShortStandaloneMonthSymbols:

Sets the very short standalone month symbols for the receiver.

```
- (void)setVeryShortStandaloneMonthSymbols:(NSArray *)array
```

Parameters

array

An array of NSString objects that specify the very short standalone month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- veryShortStandaloneMonthSymbols (page 270)
- setMonthSymbols: (page 255)
- setShortMonthSymbols: (page 256)
- setVeryShortMonthSymbols: (page 262)
- setStandaloneMonthSymbols: (page 259)
- setShortStandaloneMonthSymbols: (page 257)

Declared In
NSDateFormatter.h

setVeryShortStandaloneWeekdaySymbols:

Sets the very short standalone weekday symbols for the receiver.

```
- (void)setVeryShortStandaloneWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short standalone weekday symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- veryShortStandaloneWeekdaySymbols` (page 270)
- `- setWeekdaySymbols:` (page 263)
- `- setShortWeekdaySymbols:` (page 259)
- `- setVeryShortWeekdaySymbols:` (page 263)
- `- setStandaloneWeekdaySymbols:` (page 260)
- `- setShortStandaloneWeekdaySymbols:` (page 258)

Declared In

`NSDateFormatter.h`

setVeryShortWeekdaySymbols:

Sets the vert short weekday symbols for the receiver

```
- (void)setVeryShortWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the very short weekday symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- veryShortWeekdaySymbols` (page 271)
- `- setWeekdaySymbols:` (page 263)
- `- setShortWeekdaySymbols:` (page 259)
- `- setStandaloneWeekdaySymbols:` (page 260)
- `- setShortStandaloneWeekdaySymbols:` (page 258)
- `- setVeryShortStandaloneWeekdaySymbols:` (page 263)

Declared In

`NSDateFormatter.h`

setWeekdaySymbols:

Sets the weekday symbols for the receiver.

```
- (void)setWeekdaySymbols:(NSArray *)array
```

Parameters

array

An array of `NSString` objects that specify the weekday symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `weekdaySymbols` (page 271)
- `setShortWeekdaySymbols:` (page 259)
- `setVeryShortWeekdaySymbols:` (page 263)
- `setStandaloneWeekdaySymbols:` (page 260)
- `setShortStandaloneWeekdaySymbols:` (page 258)
- `setVeryShortStandaloneWeekdaySymbols:` (page 263)

Declared In

`NSDateFormatter.h`

shortMonthSymbols

Returns the array of short month symbols for the receiver.

```
- (NSArray *)shortMonthSymbols
```

Return Value

An array containing `NSString` objects representing the short month symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setShortMonthSymbols:` (page 256)
- `monthSymbols` (page 249)
- `veryShortMonthSymbols` (page 269)
- `standaloneMonthSymbols` (page 267)
- `shortStandaloneMonthSymbols` (page 265)
- `veryShortStandaloneMonthSymbols` (page 270)

Declared In

`NSDateFormatter.h`

shortQuarterSymbols

Returns the short quarter symbols for the receiver.

```
- (NSArray *)shortQuarterSymbols
```

Return Value

An array containing `NSString` objects representing the short quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setShortQuarterSymbols:` (page 257)
- `quarterSymbols` (page 250)
- `standaloneQuarterSymbols` (page 267)
- `shortStandaloneQuarterSymbols` (page 265)

Declared In
`NSDateFormatter.h`

shortStandaloneMonthSymbols

Returns the short standalone month symbols for the receiver.

```
- (NSArray *)shortStandaloneMonthSymbols
```

Return Value
An array of `NSString` objects that specify the short standalone month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setShortStandaloneMonthSymbols:` (page 257)
- `monthSymbols` (page 249)
- `shortMonthSymbols` (page 264)
- `veryShortMonthSymbols` (page 269)
- `standaloneMonthSymbols` (page 267)
- `veryShortStandaloneMonthSymbols` (page 270)

Declared In
`NSDateFormatter.h`

shortStandaloneQuarterSymbols

Returns the short standalone quarter symbols for the receiver.

```
- (NSArray *)shortStandaloneQuarterSymbols
```

Return Value
An array containing `NSString` objects representing the short standalone quarter symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setShortStandaloneQuarterSymbols:` (page 258)
- `quarterSymbols` (page 250)

- shortQuarterSymbols (page 264)
- standaloneQuarterSymbols (page 267)

Declared In
NSDateFormatter.h

shortStandaloneWeekdaySymbols

Returns the array of short standalone weekday symbols for the receiver.

```
- (NSArray *)shortStandaloneWeekdaySymbols
```

Return Value
An array of NSString objects that specify the short standalone weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setShortStandaloneWeekdaySymbols: (page 258)
- weekdaySymbols (page 271)
- shortWeekdaySymbols (page 266)
- veryShortWeekdaySymbols (page 271)
- standaloneWeekdaySymbols (page 267)
- veryShortStandaloneWeekdaySymbols (page 270)

Declared In
NSDateFormatter.h

shortWeekdaySymbols

Returns the array of short weekday symbols for the receiver.

```
- (NSArray *)shortWeekdaySymbols
```

Return Value
An array of NSString objects that specify the short weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setShortWeekdaySymbols: (page 259)
- weekdaySymbols (page 271)
- veryShortWeekdaySymbols (page 271)
- standaloneWeekdaySymbols (page 267)
- shortStandaloneWeekdaySymbols (page 266)
- veryShortStandaloneWeekdaySymbols (page 270)

Declared In
NSDateFormatter.h

standaloneMonthSymbols

Returns the standalone month symbols for the receiver.

```
- (NSArray *)standaloneMonthSymbols
```

Return Value

An array of `NSString` objects that specify the standalone month symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `monthSymbols` (page 249)
- `setStandaloneMonthSymbols:` (page 259)
- `shortMonthSymbols` (page 264)
- `veryShortMonthSymbols` (page 269)
- `shortStandaloneMonthSymbols` (page 265)
- `veryShortStandaloneMonthSymbols` (page 270)

Declared In

`NSDateFormatter.h`

standaloneQuarterSymbols

Returns the standalone quarter symbols for the receiver.

```
- (NSArray *)standaloneQuarterSymbols
```

Return Value

An array containing `NSString` objects representing the standalone quarter symbols for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setStandaloneQuarterSymbols:` (page 260)
- `quarterSymbols` (page 250)
- `shortQuarterSymbols` (page 264)
- `shortStandaloneQuarterSymbols` (page 265)

Declared In

`NSDateFormatter.h`

standaloneWeekdaySymbols

Returns the array of standalone weekday symbols for the receiver.

```
- (NSArray *)standaloneWeekdaySymbols
```

Return Value

An array of `NSString` objects that specify the standalone weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setStandaloneWeekdaySymbols:` (page 260)
- `weekdaySymbols` (page 271)
- `shortWeekdaySymbols` (page 266)
- `veryShortWeekdaySymbols` (page 271)
- `shortStandaloneWeekdaySymbols` (page 266)
- `veryShortStandaloneWeekdaySymbols` (page 270)

Declared In
`NSDateFormatter.h`

stringFromDate:

Returns a string representation of a given date formatted using the receiver's current settings.

```
- (NSString *)stringFromDate:(NSDate *)date
```

Parameters

date
The date to format.

Return Value
A string representation of *date* formatted using the receiver's current settings.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `dateFromString:` (page 244)

Declared In
`NSDateFormatter.h`

timeStyle

Returns the time style of the receiver.

```
- (NSDateFormatterStyle)timeStyle
```

Return Value
The time style of the receiver. For possible values, see NSDateFormatterStyle (page 272).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTimeStyle:` (page 261)

Declared In
`NSDateFormatter.h`

timeZone

Returns the time zone for the receiver.

```
- (NSTimeZone *)timeZone
```

Return Value
The time zone for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTimeZone:` (page 261)

Declared In
`NSDateFormatter.h`

twoDigitStartDate

Returns the earliest date that can be denoted by a two-digit year specifier.

```
- (NSDate *)twoDigitStartDate
```

Return Value
The earliest date that can be denoted by a two-digit year specifier.

Discussion
If the two-digit start date is set to January 6, 1976, then “January 1, 76” is interpreted as New Year's Day in 2076, whereas “February 14, 76” is interpreted as Valentine's Day in 1976.

The default date is December 31, 1949.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTwoDigitStartDate:` (page 261)

Declared In
`NSDateFormatter.h`

veryShortMonthSymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)veryShortMonthSymbols
```

Return Value
An array of `NSString` objects that specify the very short month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setVeryShortMonthSymbols:` (page 262)
- `monthSymbols` (page 249)
- `shortMonthSymbols` (page 264)
- `standaloneMonthSymbols` (page 267)
- `shortStandaloneMonthSymbols` (page 265)
- `veryShortStandaloneMonthSymbols` (page 270)

Declared In
`NSDateFormatter.h`

veryShortStandaloneMonthSymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)veryShortStandaloneMonthSymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone month symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setVeryShortStandaloneMonthSymbols:` (page 262)
- `monthSymbols` (page 249)
- `shortMonthSymbols` (page 264)
- `veryShortMonthSymbols` (page 269)
- `standaloneMonthSymbols` (page 267)
- `shortStandaloneMonthSymbols` (page 265)

Declared In
`NSDateFormatter.h`

veryShortStandaloneWeekdaySymbols

Returns the array of very short standalone weekday symbols for the receiver.

```
- (NSArray *)veryShortStandaloneWeekdaySymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setShortStandaloneWeekdaySymbols:` (page 258)
- `weekdaySymbols` (page 271)
- `shortWeekdaySymbols` (page 266)
- `veryShortWeekdaySymbols` (page 271)

- standaloneWeekdaySymbols (page 267)
- shortStandaloneWeekdaySymbols (page 266)

Declared In
NSDateFormatter.h

veryShortWeekdaySymbols

Returns the array of very short weekday symbols for the receiver.

```
- (NSArray *)veryShortWeekdaySymbols
```

Return Value
An array of NSString objects that specify the very short weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setVeryShortWeekdaySymbols: (page 263)
- weekdaySymbols (page 271)
- shortWeekdaySymbols (page 266)
- standaloneWeekdaySymbols (page 267)
- shortStandaloneWeekdaySymbols (page 266)
- veryShortStandaloneWeekdaySymbols (page 270)

Declared In
NSDateFormatter.h

weekdaySymbols

Returns the array of weekday symbols for the receiver.

```
- (NSArray *)weekdaySymbols
```

Return Value
An array of NSString objects that specify the weekday symbols for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setWeekdaySymbols: (page 263)
- shortWeekdaySymbols (page 266)
- veryShortWeekdaySymbols (page 271)
- standaloneWeekdaySymbols (page 267)
- shortStandaloneWeekdaySymbols (page 266)
- veryShortStandaloneWeekdaySymbols (page 270)

Declared In
NSDateFormatter.h

Constants

NSDateFormatterStyle

The following constants specify predefined date and time format styles.

```
typedef enum {
    NSDateFormatterNoStyle     = kCFDateFormatterNoStyle,
    NSDateFormatterShortStyle  = kCFDateFormatterShortStyle,
    NSDateFormatterMediumStyle = kCFDateFormatterMediumStyle,
    NSDateFormatterLongStyle   = kCFDateFormatterLongStyle,
    NSDateFormatterFullStyle   = kCFDateFormatterFullStyle
} NSDateFormatterStyle;
```

Constants

`NSDateFormatterNoStyle`

Specifies no style.

Equal to `kCFDateFormatterNoStyle`.

Available in iPhone OS 2.0 and later.

Declared in `NSDateFormatter.h`

`NSDateFormatterShortStyle`

Specifies a short style, typically numeric only, such as “11/23/37” or “3:30pm”.

Equal to `kCFDateFormatterShortStyle`.

Available in iPhone OS 2.0 and later.

Declared in `NSDateFormatter.h`

`NSDateFormatterMediumStyle`

Specifies a medium style, typically with abbreviated text, such as “Nov 23, 1937”.

Equal to `kCFDateFormatterMediumStyle`.

Available in iPhone OS 2.0 and later.

Declared in `NSDateFormatter.h`

`NSDateFormatterLongStyle`

Specifies a long style, typically with full text, such as “November 23, 1937” or “3:30:32pm”.

Equal to `kCFDateFormatterLongStyle`.

Available in iPhone OS 2.0 and later.

Declared in `NSDateFormatter.h`

`NSDateFormatterFullStyle`

Specifies a full style with complete details, such as “Tuesday, April 12, 1952 AD” or “3:30:42pm PST”.

Equal to `kCFDateFormatterFullStyle`.

Available in iPhone OS 2.0 and later.

Declared in `NSDateFormatter.h`

Discussion

The format for these date and time styles is not exact because they depend on the locale, user preference settings, and the operating system version. Do not use these constants if you want an exact format.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDateFormatter.h

NSDateFormatterBehavior

Constants that specify the behavior NSDateFormatter should exhibit.

```
typedef enum {
    NSDateFormatterBehaviorDefault = 0,
    NSDateFormatterBehavior10_0    = 1000,
    NSDateFormatterBehavior10_4    = 1040,
} NSDateFormatterBehavior;
```

Constants

NSDateFormatterBehaviorDefault

Specifies default formatting behavior.

Available in iPhone OS 2.0 and later.

Declared in NSDateFormatter.h

NSDateFormatterBehavior10_0

Specifies formatting behavior equivalent to that in Mac OS X 10.0.

Available in iPhone OS 2.0 and later.

Declared in NSDateFormatter.h

NSDateFormatterBehavior10_4

Specifies formatting behavior equivalent for Mac OS X 10.4.

Available in iPhone OS 2.0 and later.

Declared in NSDateFormatter.h

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDateFormatter.h

NSDecimalNumber Class Reference

Inherits from:	NSNumber : NSValue : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDecimalNumber.h
Companion guide:	Number and Value Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSDecimalNumber`, an immutable subclass of `NSNumber`, provides an object-oriented wrapper for doing base-10 arithmetic. An instance can represent any number that can be expressed as `mantissa x 10^exponent` where mantissa is a decimal integer up to 38 digits long, and exponent is an integer from –128 through 127.

Tasks

Creating a Decimal Number

`+ decimalNumberWithDecimal:` (page 278)
Creates and returns an `NSDecimalNumber` object equivalent to a given `NSDecimal` structure.

`+ decimalNumberWithMantissa:exponent:isNegative:` (page 279)
Creates and returns an `NSDecimalNumber` object equivalent to the number specified by the arguments.

`+ decimalNumberWithString:` (page 279)
Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string.

`+ decimalNumberWithString:locale:` (page 280)
Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string, interpreted using a given locale.

`+ one` (page 282)
Returns an `NSDecimalNumber` object equivalent to the number 1.0.

`+ zero` (page 283)
Returns an `NSDecimalNumber` object equivalent to the number 0.0.

`+ notANumber` (page 282)
Returns an `NSDecimalNumber` object that specifies no number.

Initializing a Decimal Number

`- initWithDecimal:` (page 290)
Returns an `NSDecimalNumber` object initialized to represent a given decimal.

`- initWithMantissa:exponent:isNegative:` (page 290)
Returns an `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

`- initWithString:` (page 291)
Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string.

`- initWithString:locale:` (page 291)
Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string, interpreted using a given locale.

Performing Arithmetic

`- decimalNumberByAdding:` (page 284)
Returns a new `NSDecimalNumber` object whose value is the sum of the receiver and another given `NSDecimalNumber` object.

`- decimalNumberBySubtracting:` (page 288)
Returns a new `NSDecimalNumber` object whose value is that of another given `NSDecimalNumber` object subtracted from the value of the receiver.

- `decimalNumberByMultiplyingBy:` (page 286)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver multiplied by that of another given `NSDecimalNumber` object.
- `decimalNumberByDividingBy:` (page 285)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver divided by that of another given `NSDecimalNumber` object.
- `decimalNumberByRaisingToPower:` (page 287)
  Returns a new `NSDecimalNumber` object whose value is the value of the receiver raised to a given power.
- `decimalNumberByMultiplyingByPowerOf10:` (page 286)
  Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.
- `decimalNumberByAdding:withBehavior:` (page 284)
  Adds *decimalNumber* to the receiver and returns the sum, a newly created `NSDecimalNumber` object.
- `decimalNumberBySubtracting:withBehavior:` (page 289)
  Subtracts *decimalNumber* from the receiver and returns the difference, a newly created `NSDecimalNumber` object.
- `decimalNumberByMultiplyingBy:withBehavior:` (page 286)
  Multiplies the receiver by *decimalNumber* and returns the product, a newly created `NSDecimalNumber` object.
- `decimalNumberByDividingBy:withBehavior:` (page 285)
  Divides the receiver by *decimalNumber* and returns the quotient, a newly created `NSDecimalNumber` object.
- `decimalNumberByRaisingToPower:withBehavior:` (page 288)
  Raises the receiver to *power* and returns the result, a newly created `NSDecimalNumber` object.
- `decimalNumberByMultiplyingByPowerOf10:withBehavior:` (page 287)
  Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

Rounding Off

- `decimalNumberByRoundingAccordingToBehavior:` (page 288)
  Rounds the receiver off in the way specified by *behavior* and returns the result, a newly created `NSDecimalNumber` object.

Accessing the Value

- `decimalValue` (page 289)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 290)
  Returns the approximate value of the receiver as a `double`.
- `descriptionWithLocale:` (page 289)
  Returns a string, specified according to a given locale, that represents the contents of the receiver.

- objCType (page 292)
  Returns a C string containing the Objective-C type of the data contained in the receiver, which for an NSDecimalNumber object is always "d" (for double).

Managing Behavior

+ defaultBehavior (page 281)
  Returns the way arithmetic methods, like decimalNumberByAdding: (page 284), round off and handle error conditions.

+ setDefaultBehavior: (page 283)
  Specifies the way that arithmetic methods, like decimalNumberByAdding: (page 284), round off and handle error conditions.

Comparing Decimal Numbers

- compare: (page 283)
  Returns an NSComparisonResult value that indicates the numerical ordering of the receiver and another given NSDecimalNumber object.

Getting Maximum and Minimum Possible Values

+ maximumDecimalNumber (page 281)
  Returns the largest possible value of an NSDecimalNumber object.

+ minimumDecimalNumber (page 282)
  Returns the smallest possible value of an NSDecimalNumber object.

Class Methods

decimalNumberWithDecimal:

Creates and returns an NSDecimalNumber object equivalent to a given NSDecimal structure.

```
+ (NSDecimalNumber *)decimalNumberWithDecimal:(NSDecimal)decimal
```

Parameters

decimal
  An NSDecimal structure that specifies the value for the new decimal number object.

Return Value

An NSDecimalNumber object equivalent to *decimal*.

Discussion

You can initialize *decimal* programmatically or generate it using the NSScanner method, scanDecimal: (page 905)

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

decimalNumberWithMantissa:exponent:isNegative:

Creates and returns an NSDecimalNumber object equivalent to the number specified by the arguments.

```
+ (NSDecimalNumber *)decimalNumberWithMantissa:(unsigned long long)mantissa
    exponent:(short)exponent isNegative:(BOOL)isNegative
```

Parameters
mantissa
The mantissa for the new decimal number object.

exponent
The exponent for the new decimal number object.

isNegative
A Boolean value that specifies whether the sign of the number is negative.

Discussion
The arguments express a number in a kind of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is -12.345, it is expressed as 12345x10^-3—*mantissa* is 12345; *exponent* is -3; and *isNegative* is YES, as illustrated by the following example.

```
NSDecimalNumber *number = [NSDecimalNumber decimalNumberWithMantissa:12345
                                                exponent:-3
                                                isNegative:YES];
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

decimalNumberWithString:

Creates and returns an NSDecimalNumber object whose value is equivalent to that in a given numeric string.

```
+ (NSDecimalNumber *)decimalNumberWithString:(NSString *)numericString
```

Parameters
numericString
A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single NSDecimalSeparator to divide the fractional from the integral part of the number.

Return Value
An NSDecimalNumber object whose value is equivalent to *numericString*.

Discussion
Whether the `NSDecimalSeparator` is a period (as is used, for example, in the United States) or a comma (as is used, for example, in France) depends on the default locale.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ decimalNumberWithString:locale:` (page 280)

Declared In
`NSDecimalNumber.h`

decimalNumberWithString:locale:

Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string, interpreted using a given locale.

```
+ (NSDecimalNumber *)decimalNumberWithString:(NSString *)numericString
    locale:(NSDictionary *)locale
```

Parameters
numericString

A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number.

locale

A dictionary that defines the locale (specifically the `NSDecimalSeparator`) to use to interpret the number in *numericString*.

Return Value
An `NSDecimalNumber` object whose value is equivalent to *numericString*.

Discussion
The *locale* parameter determines whether the `NSDecimalSeparator` is a period (as is used, for example, in the United States) or a comma (as is used, for example, in France).

The following strings show examples of acceptable values for *numericString*:

"2500.6" (or "2500,6", depending on locale)
"–2500.6" (or "–2500.6")
"–2.5006e3" (or "–2,5006e3")
"–2.5006E3" (or "–2,5006E3")

The following strings are unacceptable:

"2,500.6"
"2500 3/5"
"2.5006x10e3"
"two thousand five hundred and six tenths"

Availability
Available in iPhone OS 2.0 and later.

See Also
+ decimalNumberWithString: (page 279)

Declared In
NSDecimalNumber.h

defaultBehavior

Returns the way arithmetic methods, like decimalNumberByAdding: (page 284), round off and handle error conditions.

```
+ (id < NSDecimalNumberBehaviors >)defaultBehavior
```

Discussion
By default, the arithmetic methods use the NSRoundPlain behavior; that is, the methods round to the closest possible return value. The methods assume your need for precision does not exceed 38 significant digits and raise exceptions when they try to divide by 0 or produce a number too big or too small to be represented.

If this default behavior doesn't suit your application, you should use methods that let you specify the behavior, like decimalNumberByAdding:withBehavior: (page 284). If you find yourself using a particular behavior consistently, you can specify a different default behavior with setDefaultBehavior: (page 283).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

maximumDecimalNumber

Returns the largest possible value of an NSDecimalNumber object.

```
+ (NSDecimalNumber *)maximumDecimalNumber
```

Return Value
The largest possible value of an NSDecimalNumber object.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ minimumDecimalNumber (page 282)

Declared In
NSDecimalNumber.h

minimumDecimalNumber

Returns the smallest possible value of an `NSDecimalNumber` object.

```
+ (NSDecimalNumber *)minimumDecimalNumber
```

Return Value
The smallest possible value of an `NSDecimalNumber` object.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ maximumDecimalNumber` (page 281)

Declared In
`NSDecimalNumber.h`

notANumber

Returns an `NSDecimalNumber` object that specifies no number.

```
+ (NSDecimalNumber *)notANumber
```

Return Value
An `NSDecimalNumber` object that specifies no number.

Discussion
Any arithmetic method receiving `notANumber` as an argument returns `notANumber`.

This value can be a useful way of handling non-numeric data in an input file. This method can also be a useful response to calculation errors. For more information on calculation errors, see the `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1252) method description in the `NSDecimalNumberBehaviors` protocol specification.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

one

Returns an `NSDecimalNumber` object equivalent to the number 1.0.

```
+ (NSDecimalNumber *)one
```

Return Value
An `NSDecimalNumber` object equivalent to the number 1.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ zero` (page 283)

Declared In
NSDecimalNumber.h

setDefaultBehavior:

Specifies the way that arithmetic methods, like `decimalNumberByAdding:` (page 284), round off and handle error conditions.

```
+ (void)setDefaultBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior must conform to the `NSDecimalNumberBehaviors` protocol.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

zero

Returns an `NSDecimalNumber` object equivalent to the number 0.0.

```
+ (NSDecimalNumber *)zero
```

Return Value
An `NSDecimalNumber` object equivalent to the number 0.0.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ one` (page 282)

Declared In
NSDecimalNumber.h

Instance Methods

compare:

Returns an `NSComparisonResult` value that indicates the numerical ordering of the receiver and another given `NSDecimalNumber` object.

```
- (NSComparisonResult)compare:(NSNumber *)decimalNumber
```

Parameters

decimalNumber

The number with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` if the value of *decimalNumber* is greater than the receiver; `NSOrderedSame` if they're equal; and `NSOrderedDescending` if the value of *decimalNumber* is less than the receiver.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

decimalNumberByAdding:

Returns a new `NSDecimalNumber` object whose value is the sum of the receiver and another given `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByAdding:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber

The number to add to the receiver.

Return Value

A new `NSDecimalNumber` object whose value is the sum of the receiver and *decimalNumber*.

Discussion

This method uses the default behavior when handling calculation errors and rounding.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- decimalNumberByAdding:withBehavior:` (page 284)

`+ defaultBehavior` (page 281)

Declared In

`NSDecimalNumber.h`

decimalNumberByAdding:withBehavior:

Adds *decimalNumber* to the receiver and returns the sum, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByAdding:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion

behavior specifies the handling of calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

decimalNumberByDividingBy:

Returns a new NSDecimalNumber object whose value is the value of the receiver divided by that of another given NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByDividingBy:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber

The number by which to divide the receiver.

Return Value
A new NSDecimalNumber object whose value is the value of the receiver divided by *decimalNumber*.

Discussion
This method uses the default behavior when handling calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decimalNumberByDividingBy:withBehavior: (page 285)
+ defaultBehavior (page 281)

Declared In
NSDecimalNumber.h

decimalNumberByDividingBy:withBehavior:

Divides the receiver by *decimalNumber* and returns the quotient, a newly created NSDecimalNumber object.

```
- (NSDecimalNumber *)decimalNumberByDividingBy:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

decimalNumberByMultiplyingBy:

Returns a new `NSDecimalNumber` object whose value is the value of the receiver multiplied by that of another given `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingBy:(NSDecimalNumber *)decimalNumber
```

Parameters

decimalNumber

The number by which to multiply the receiver.

Return Value

A new `NSDecimalNumber` object whose value is *decimalNumber* multiplied by the receiver.

Discussion

This method uses the default behavior when handling calculation errors and when rounding.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `decimalNumberByMultiplyingBy:withBehavior:` (page 286)
+ `defaultBehavior` (page 281)

Declared In

`NSDecimalNumber.h`

decimalNumberByMultiplyingBy:withBehavior:

Multiplies the receiver by *decimalNumber* and returns the product, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingBy:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion

behavior specifies the handling of calculation errors and rounding.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

decimalNumberByMultiplyingByPowerOf10:

Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingByPowerOf10:(short)power
```

Discussion

This method uses the default behavior when handling calculation errors and when rounding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decimalNumberByMultiplyingByPowerOf10:withBehavior:` (page 287)
+ `defaultBehavior` (page 281)

Declared In
`NSDecimalNumber.h`

decimalNumberByMultiplyingByPowerOf10:withBehavior:
Multiplies the receiver by 10^*power* and returns the product, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByMultiplyingByPowerOf10:(short)power
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

decimalNumberByRaisingToPower:
Returns a new `NSDecimalNumber` object whose value is the value of the receiver raised to a given power.

```
- (NSDecimalNumber *)decimalNumberByRaisingToPower:(NSUInteger)power
```

Parameters
power
The power to which to raise the receiver.

Return Value
A new `NSDecimalNumber` object whose value is the value of the receiver raised to the power *power*.

Discussion
This method uses the default behavior when handling calculation errors and when rounding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decimalNumberByRaisingToPower:withBehavior:` (page 288)
+ `defaultBehavior` (page 281)

Declared In
`NSDecimalNumber.h`

decimalNumberByRaisingToPower:withBehavior:

Raises the receiver to *power* and returns the result, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByRaisingToPower:(NSUInteger)power withBehavior:(id
    < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

decimalNumberByRoundingAccordingToBehavior:

Rounds the receiver off in the way specified by *behavior* and returns the result, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberByRoundingAccordingToBehavior:(id <
    NSDecimalNumberBehaviors >)behavior
```

Discussion
For a description of the different ways of rounding, see the `roundingMode` (page 745) method in the `NSDecimalNumberBehaviors` protocol specification.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

decimalNumberBySubtracting:

Returns a new `NSDecimalNumber` object whose value is that of another given `NSDecimalNumber` object subtracted from the value of the receiver.

```
- (NSDecimalNumber *)decimalNumberBySubtracting:(NSDecimalNumber *)decimalNumber
```

Parameters
decimalNumber

The number to subtract from the receiver.

Return Value
A new `NSDecimalNumber` object whose value is *decimalNumber* subtracted from the receiver.

Discussion
This method uses the default behavior when handling calculation errors and when rounding.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `decimalNumberBySubtracting:withBehavior:` (page 289)
+ `defaultBehavior` (page 281)

Declared In
`NSDecimalNumber.h`

decimalNumberBySubtracting:withBehavior:

Subtracts *decimalNumber* from the receiver and returns the difference, a newly created `NSDecimalNumber` object.

```
- (NSDecimalNumber *)decimalNumberBySubtracting:(NSDecimalNumber *)decimalNumber
    withBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
behavior specifies the handling of calculation errors and rounding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

decimalValue

Returns the receiver's value, expressed as an `NSDecimal` structure.

```
- (NSDecimal)decimalValue
```

Return Value
The receiver's value, expressed as an `NSDecimal` structure.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

descriptionWithLocale:

Returns a string, specified according to a given locale, that represents the contents of the receiver.

```
- (NSString *)descriptionWithLocale:(NSDictionary *)locale
```

Parameters
locale
A dictionary that defines the locale (specifically the `NSDecimalSeparator`) to use to generate the returned string.

Return Value
A string that represents the contents of the receiver, according to *locale*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

doubleValue

Returns the approximate value of the receiver as a `double`.

```
- (double)doubleValue
```

Return Value
The approximate value of the receiver as a `double`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

initWithDecimal:

Returns an `NSDecimalNumber` object initialized to represent a given decimal.

```
- (id)initWithDecimal:(NSDecimal)decimal
```

Parameters
decimal
The value of the new object.

Return Value
An `NSDecimalNumber` object initialized to represent *decimal*.

Discussion
This method is the designated initializer for `NSDecimalNumber`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimalNumber.h

initWithMantissa:exponent:isNegative:

Returns an `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

```
- (id)initWithMantissa:(unsigned long long)mantissa exponent:(short)exponent
    isNegative:(BOOL)flag
```

Parameters
mantissa
The mantissa for the new decimal number object.

exponent
The exponent for the new decimal number object.

flag
A Boolean value that specifies whether the sign of the number is negative.

Return Value
An `NSDecimalNumber` object initialized using the given mantissa, exponent, and sign.

Discussion
The arguments express a number in a type of scientific notation that requires the mantissa to be an integer. So, for example, if the number to be represented is 1.23, it is expressed as 123x10^–2—*mantissa* is 123; *exponent* is –2; and *isNegative*, which refers to the sign of the mantissa, is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ decimalNumberWithMantissa:exponent:isNegative:` (page 279)

Declared In
`NSDecimalNumber.h`

initWithString:

Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string.

```
- (id)initWithString:(NSString *)numericString
```

Parameters
numericString
A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number. For a listing of acceptable and unacceptable strings, see the class method `decimalNumberWithString:locale:` (page 280).

Return Value
An `NSDecimalNumber` object initialized so that its value is equivalent to that in *numericString*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

initWithString:locale:

Returns an `NSDecimalNumber` object initialized so that its value is equivalent to that in a given numeric string, interpreted using a given locale.

```
- (id)initWithString:(NSString *)numericString locale:(NSDictionary *)locale
```

Parameters

numericString

A numeric string.

Besides digits, *numericString* can include an initial "+" or "–"; a single "E" or "e", to indicate the exponent of a number in scientific notation; and a single `NSDecimalSeparator` to divide the fractional from the integral part of the number.

locale

A dictionary that defines the locale (specifically the `NSDecimalSeparator`) to use to interpret the number in *numericString*.

Return Value

An `NSDecimalNumber` object initialized so that its value is equivalent to that in *numericString*, interpreted using *locale*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ decimalNumberWithString:locale:` (page 280)

Declared In

`NSDecimalNumber.h`

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver, which for an `NSDecimalNumber` object is always "d" (for double).

```
- (const char *)objCType
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

Constants

NSDecimalNumber Exception Names

Names of the various exceptions raised by `NSDecimalNumber` to indicate computational errors.

```
extern NSString *NSDecimalNumberExactnessException;
extern NSString *NSDecimalNumberOverflowException;
extern NSString *NSDecimalNumberUnderflowException;
extern NSString *NSDecimalNumberDivideByZeroException;
```

Constants

`NSDecimalNumberExactnessException`

The name of the exception raised if there is an exactness error.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimalNumber.h`

`NSDecimalNumberOverflowException`

The name of the exception raised on overflow.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimalNumber.h`

`NSDecimalNumberUnderflowException`

The name of the exception raised on underflow.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimalNumber.h`

`NSDecimalNumberDivideByZeroException`

The name of the exception raised on divide by zero.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimalNumber.h`

Declared In

`NSDecimalNumber.h`

NSDecimalNumberHandler Class Reference

Inherits from:	NSObject
Conforms to:	NSDecimalNumberBehaviors NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDecimalNumber.h
Companion guide:	Number and Value Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSDecimalNumberHandler` is a class that adopts the `NSDecimalNumberBehaviors` protocol. This class allows you to set the way an `NSDecimalNumber` object rounds off and handles errors, without having to create a custom class.

You can use an instance of this class as an argument to any of the `NSDecimalNumber` methods that end with `...Behavior:`. If you don't think you need special behavior, you probably don't need this class—it is likely that `NSDecimalNumber`'s default behavior will suit your needs.

For more information, see the NSDecimalNumberBehaviors protocol specification.

Adopted Protocols

NSDecimalNumberBehaviors
- roundingMode (page 1253)
- scale (page 1253)
- exceptionDuringOperation:error:leftOperand:rightOperand: (page 1252)

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

Tasks

Creating a Decimal Number Handler

+ defaultDecimalNumberHandler (page 297)
Returns the default instance of NSDecimalNumberHandler.

+ decimalNumberHandlerWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero: (page 296)
Returns an NSDecimalNumberHandler object with customized behavior.

Initializing a Decimal Number Handler

- initWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero: (page 298)
Returns an NSDecimalNumberHandler object initialized so it behaves as specified by the method's arguments.

Class Methods

decimalNumberHandlerWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:

Returns an NSDecimalNumberHandler object with customized behavior.

```
+ (id)decimalNumberHandlerWithRoundingMode:(NSRoundingMode)roundingMode
    scale:(short)scale raiseOnExactness:(BOOL)raiseOnExactness
    raiseOnOverflow:(BOOL)raiseOnOverflow raiseOnUnderflow:(BOOL)raiseOnUnderflow
     raiseOnDivideByZero:(BOOL)raiseOnDivideByZero
```

Parameters

roundingMode

The rounding mode to use. There are four possible values: `NSRoundUp`, `NSRoundDown`, `NSRoundPlain`, and `NSRoundBankers`.

scale

The number of digits a rounded value should have after its decimal point.

raiseOnExactness

If `YES`, in the event of an exactness error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method.

raiseOnOverflow

If `YES`, in the event of an overflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnUnderflow

If `YES`, in the event of an underflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnDivideByZero

If `YES`, in the event of a divide by zero error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

Return Value

An `NSDecimalNumberHandler` object with customized behavior.

Discussion

See the NSDecimalNumberBehaviors protocol specification for a complete explanation of the possible behaviors.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

defaultDecimalNumberHandler

Returns the default instance of `NSDecimalNumberHandler`.

```
+ (id)defaultDecimalNumberHandler
```

Return Value

The default instance of `NSDecimalNumberHandler`.

Discussion

This default decimal number handler rounds to the closest possible return value. It assumes your need for precision does not exceed 38 significant digits, and it raises an exception when its `NSDecimalNumber` object tries to divide by `0` or when its `NSDecimalNumber` object produces a number too big or too small to be represented.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

Instance Methods

initWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:

Returns an `NSDecimalNumberHandler` object initialized so it behaves as specified by the method's arguments.

```
- (id)initWithRoundingMode:(NSRoundingMode)roundingMode scale:(short)scale
    raiseOnExactness:(BOOL)raiseOnExactness raiseOnOverflow:(BOOL)raiseOnOverflow
     raiseOnUnderflow:(BOOL)raiseOnUnderflow
    raiseOnDivideByZero:(BOOL)raiseOnDivideByZero
```

Parameters

roundingMode

The rounding mode to use. There are four possible values: `NSRoundUp`, `NSRoundDown`, `NSRoundPlain`, and `NSRoundBankers`.

scale

The number of digits a rounded value should have after its decimal point.

raiseOnExactness

If `YES`, in the event of an exactness error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method.

raiseOnOverflow

If `YES`, in the event of an overflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnUnderflow

If `YES`, in the event of an underflow error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

raiseOnDivideByZero

If `YES`, in the event of a divide by zero error the handler will raise an exception, otherwise it will ignore the error and return control to the calling method

Return Value

An initialized `NSDecimalNumberHandler` object initialized with customized behavior. The returned object might be different than the original receiver.

Discussion

See the NSDecimalNumberBehaviors protocol specification for a complete explanation of the possible behaviors.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

NSDictionary Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSObject (`NSObject`)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDictionary.h Foundation/NSFileManager.h Foundation/NSKeyValueCoding.h
Companion guides:	Collections Programming Topics for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSDictionary` class declares the programmatic interface to objects that manage immutable associations of keys and values. Use this class or its subclass `NSMutableDictionary` when you need a convenient and efficient way to retrieve data associated with an arbitrary key. (For convenience, we use the term **dictionary** to refer to any instance of one of these classes without specifying its exact class membership.)

A key-value pair within a dictionary is called an entry. Each entry consists of one object that represents the key and a second object that is that key's value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by `isEqual:` (page 1306)). In general, a key can be any object (provided that it conforms to the `NSCopying` protocol—see below), but note that when using key-value coding the key must be a string (see Key-Value Coding Fundamentals). Neither a key nor a value can be `nil`; if you need to represent a null value in a dictionary, you should use `NSNull`.

An instance of `NSDictionary` is an immutable dictionary: you establish its entries when it's created and cannot modify them afterward. An instance of NSMutableDictionary is a mutable dictionary: you can add or delete entries at any time, and the object automatically allocates memory as needed. The dictionary classes adopt the `NSCopying` and `NSMutableCopying` protocols, making it convenient to convert a dictionary of one type to the other.

`NSDictionary` and `NSMutableDictionary` are part of a class cluster, so the objects you create with this interface are not actual instances of the these two classes. Rather, the instances belong to one of their private subclasses. Although a dictionary's class is private, its interface is public, as declared by these abstract superclasses, `NSDictionary` and `NSMutableDictionary`.

Internally, a dictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key. However, the methods defined in this cluster insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys. The methods described below take keys directly, not their hashed form.

Methods that add entries to dictionaries—whether as part of initialization (for all dictionaries) or during modification (for mutable dictionaries)—copy each key argument (keys must conform to the `NSCopying` protocol) and add the copies to the dictionary. Each corresponding value object receives a `retain` (page 1312) message to ensure that it won't be deallocated before the dictionary is through with it.

Enumeration

You can enumerate the contents of a dictionary by key or by value using the `NSEnumerator` object returned by keyEnumerator (page 325) and `objectEnumerator` (page 326) respectively. On Mac OS X v10.5 and later, `NSDictionary` supports the `NSFastEnumeration` protocol. You can use the `for...in` construct to enumerate the keys of a dictionary, as illustrated in the following example.

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", @"key3", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", @"value3",
nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
forKeys:keys];

for (id key in dictionary)
{
    NSLog(@"key: %@, value: %@", key, [dictionary objectForKey:key]);
}
```

Primitive Methods

Three primitive methods of `NSDictionary`—`count` (page 311), `objectForKey:` (page 327), and `keyEnumerator` (page 325)—provide the basis for all of the other methods in its interface. The count (page 311) method returns the number of entries in the dictionary. objectForKey: (page 327) returns the value associated with a given key. keyEnumerator (page 325) returns an object that lets you iterate through each of the keys in the dictionary. The other methods declared here operate by invoking one or more of these primitives. The non-primitive methods provide convenient ways of accessing multiple entries at once.

Descriptions and Persistence

You can use the `description...` and `writeToFile:atomically:` (page 328) methods to write a *property list representation* of a dictionary to a string or to a file, respectively. These are not intended to be used for general persistent storage of your custom data objects—see instead *Archives and Serializations Programming Guide for Cocoa*.

Toll-Free Bridging

`NSDictionary` is "toll-free bridged" with its Core Foundation counterpart, CFDictionary. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSDictionary *` parameter, you can pass in a `CFDictionaryRef`, and where you see a `CFDictionaryRef` parameter, you can pass in an `NSDictionary` instance (you cast one type to the other to suppress compiler warnings). This bridging also applies to concrete subclasses of `NSDictionary`. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

NSMutableCopying
- `- mutableCopyWithZone:` (page 1300)

NSFastEnumeration
- `- countByEnumeratingWithState:objects:count:` (page 1262)

Tasks

Creating a Dictionary

`+ dictionary` (page 305)
Creates and returns an empty dictionary.

`+ dictionaryWithContentsOfFile:` (page 305)
Creates and returns a dictionary using the keys and values found in a file specified by a given path.

`+ dictionaryWithContentsOfURL:` (page 306)
Creates and returns a dictionary using the keys and values found in a resource specified by a given URL.

`+ dictionaryWithDictionary:` (page 306)
Creates and returns a dictionary containing the keys and values from another given dictionary.

`+ dictionaryWithObject:forKey:` (page 307)
Creates and returns a dictionary containing a given key and value.

`+ dictionaryWithObjects:forKeys:` (page 307)
Creates and returns a dictionary containing entries constructed from the contents of an array of keys and an array of values.

`+ dictionaryWithObjects:forKeys:count:` (page 308)
Creates and returns a dictionary containing *`count`* objects from the *`objects`* array.

`+ dictionaryWithObjectsAndKeys:` (page 309)
Creates and returns a dictionary containing entries constructed from the specified set of values and keys.

Initializing an NSDictionary Instance

`- initWithContentsOfFile:` (page 320)
Initializes a newly allocated dictionary using the keys and values found in a file at a given path.

`- initWithContentsOfURL:` (page 321)
Initializes a newly allocated dictionary using the keys and values found at a given URL.

`- initWithDictionary:` (page 321)
Initializes a newly allocated dictionary by placing in it the keys and values contained in another given dictionary.

`- initWithDictionary:copyItems:` (page 322)
Initializes a newly allocated dictionary using the objects contained in another given dictionary.

`- initWithObjects:forKeys:` (page 322)
Initializes a newly allocated dictionary with entries constructed from the contents of the *`objects`* and *`keys`* arrays.

`- initWithObjects:forKeys:count:` (page 323)
Initializes a newly allocated dictionary with *`count`* entries.

- initWithObjectsAndKeys: (page 324)
  Initializes a newly allocated dictionary with entries constructed from the specified set of values and keys.

Counting Entries

- count (page 311)
  Returns the number of entries in the receiver.

Comparing Dictionaries

- isEqualToDictionary: (page 324)
  Returns a Boolean value that indicates whether the contents of the receiver are equal to the contents of another given dictionary.

Accessing Keys and Values

- allKeys (page 309)
  Returns a new array containing the receiver's keys.
- allKeysForObject: (page 310)
  Returns a new array containing the keys corresponding to all occurrences of a given object in the receiver.
- allValues (page 310)
  Returns a new array containing the receiver's values.
- getObjects:andKeys: (page 320)
  Returns by reference C arrays of the keys and values in the receiver.
- keyEnumerator (page 325)
  Returns an enumerator object that lets you access each key in the receiver.
- keysSortedByValueUsingSelector: (page 325)
  Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.
- objectEnumerator (page 326)
  Returns an enumerator object that lets you access each value in the receiver.
- objectForKey: (page 327)
  Returns the value associated with a given key.
- objectsForKeys:notFoundMarker: (page 327)
  Returns the set of objects from the receiver that corresponds to the specified *keys* as an NSArray.
- valueForKey: (page 328)
  Returns the value associated with a given key.

Storing Dictionaries

- writeToFile:atomically: (page 328)
  Writes a property list representation of the contents of the receiver to a given path.

- writeToURL:atomically: (page 329)
  Writes a property list representation of the contents of the receiver to a given URL.

Accessing File Attributes

- fileCreationDate (page 313)
  Returns the value for the NSFileCreationDate key.
- fileExtensionHidden (page 314)
  Returns the value for the NSFileExtensionHidden key.
- fileGroupOwnerAccountID (page 314)
  Returns the value for the NSFileGroupOwnerAccountID key.
- fileGroupOwnerAccountName (page 314)
  Returns the value for the NSFileGroupOwnerAccountName key.
- fileHFSCreatorCode (page 315)
  Returns the value for the NSFileHFSCreatorCode key.
- fileHFSTypeCode (page 315)
  Returns the value for the NSFileHFSTypeCode key.
- fileIsAppendOnly (page 315)
  Returns the value for the NSFileAppendOnly key.
- fileIsImmutable (page 316)
  Returns the value for the NSFileImmutable key.
- fileModificationDate (page 316)
  Returns the value for the key NSFileModificationDate.
- fileOwnerAccountID (page 316)
  Returns the value for the NSFileOwnerAccountID key.
- fileOwnerAccountName (page 317)
  Returns the value for the key NSFileOwnerAccountName.
- filePosixPermissions (page 317)
  Returns the value for the key NSFilePosixPermissions.
- fileSize (page 318)
  Returns the value for the key NSFileSize.
- fileSystemFileNumber (page 318)
  Returns the value for the key NSFileSystemFileNumber.
- fileSystemNumber (page 319)
  Returns the value for the key NSFileSystemNumber.
- fileType (page 319)
  Returns the value for the key NSFileType.

Creating a Description

- description (page 311)
  Returns a string that represents the contents of the receiver, formatted as a property list.

- descriptionInStringsFileFormat (page 312)
  Returns a string that represents the contents of the receiver, formatted in .strings file format.
- descriptionWithLocale: (page 312)
  Returns a string object that represents the contents of the receiver, formatted as a property list.
- descriptionWithLocale:indent: (page 312)
  Returns a string object that represents the contents of the receiver, formatted as a property list.

Class Methods

dictionary

Creates and returns an empty dictionary.

```
+ (id)dictionary
```

Return Value
A new empty dictionary.

Discussion
This method is declared primarily for use with mutable subclasses of NSDictionary.

If you don't want a temporary object, you can also create an empty dictionary using alloc... and init.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDictionary.h

dictionaryWithContentsOfFile:

Creates and returns a dictionary using the keys and values found in a file specified by a given path.

```
+ (id)dictionaryWithContentsOfFile:(NSString *)path
```

Parameters
path
  A full or relative pathname. The file identified by *path* must contain a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary). For more details, see *Property List Programming Guide for Cocoa*.

Return Value
A new dictionary that contains the dictionary at *path*, or nil if there is a file error or if the contents of the file are an invalid representation of a dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithContentsOfFile: (page 320)

Declared In
NSDictionary.h

dictionaryWithContentsOfURL:

Creates and returns a dictionary using the keys and values found in a resource specified by a given URL.

```
+ (id)dictionaryWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

An URL that identifies a resource containing a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary). For more details, see *Property List Programming Guide for Cocoa*.

Return Value
A new dictionary that contains the dictionary at *aURL*, or nil if there is an error or if the contents of the resource are an invalid representation of a dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithContentsOfURL: (page 321)

Declared In
NSDictionary.h

dictionaryWithDictionary:

Creates and returns a dictionary containing the keys and values from another given dictionary.

```
+ (id)dictionaryWithDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing keys and values for the new dictionary.

Return Value
A new dictionary containing the keys and values found in *otherDictionary*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithDictionary: (page 321)

Declared In
NSDictionary.h

dictionaryWithObject:forKey:

Creates and returns a dictionary containing a given key and value.

```
+ (id)dictionaryWithObject:(id)anObject forKey:(id)aKey
```

Parameters

anObject

The value corresponding to *aKey*.

aKey

The key for *anObject*.

Return Value

A new dictionary containing a single object, *anObject*, for a single key, *aKey*.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ dictionaryWithObjects:forKeys: (page 307)
+ dictionaryWithObjects:forKeys:count: (page 308)
+ dictionaryWithObjectsAndKeys: (page 309)

Declared In

NSDictionary.h

dictionaryWithObjects:forKeys:

Creates and returns a dictionary containing entries constructed from the contents of an array of keys and an array of values.

```
+ (id)dictionaryWithObjects:(NSArray *)objects forKeys:(NSArray *)keys
```

Parameters

objects

An array containing the values for the new dictionary. Each value object receives a retain (page 1312) message before being added to the dictionary.

keys

An array containing the keys for the new dictionary. Each key is copied (using copyWithZone: (page 1250); keys must conform to the NSCopying protocol), and the copy is added to the dictionary.

Return Value

A new dictionary containing entries constructed from the contents of *objects* and *keys*.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An NSInvalidArgumentException is raised if objects and keys don't have the same number of elements.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithObjects:forKeys: (page 322)
+ dictionaryWithObject:forKey: (page 307)
+ dictionaryWithObjects:forKeys:count: (page 308)
+ dictionaryWithObjectsAndKeys: (page 309)

Declared In

NSDictionary.h

dictionaryWithObjects:forKeys:count:

Creates and returns a dictionary containing *count* objects from the *objects* array.

```
+ (id)dictionaryWithObjects:(id *)objects forKeys:(id *)keys count:(NSUInteger)count
```

Parameters

objects

A C array of values for the new dictionary. Each value object receives a retain (page 1312) message before being added to the new dictionary.

keys

A C array of keys for the new dictionary. Each key is copied (using copyWithZone: (page 1250); keys must conform to the NSCopying protocol), and the copy is added to the new dictionary.

count

The number of elements to use from the *keys* and *objects* arrays. *count* must not exceed the number of elements in *objects* or *keys*.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An NSInvalidArgumentException is raised if a key or value object is nil.

The following code fragment illustrates how to create a dictionary that associates the alphabetic characters with their ASCII values:

```
static const NSInteger N_ENTRIES = 26;
NSDictionary *asciiDict;
NSString *keyArray[N_ENTRIES];
NSNumber *valueArray[N_ENTRIES];
NSInteger i;

for (i = 0; i < N_ENTRIES; i++)
{
    char charValue = 'a' + i;
    keyArray[i] = [NSString stringWithFormat:@"%c", charValue];
    valueArray[i] = [NSNumber numberWithChar:charValue];
}

asciiDict = [NSDictionary dictionaryWithObjects:(id *)valueArray
                          forKeys:(id *)keyArray count:N_ENTRIES];
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithObjects:forKeys:count: (page 323)

+ dictionaryWithObject:forKey: (page 307)
+ dictionaryWithObjects:forKeys: (page 307)
+ dictionaryWithObjectsAndKeys: (page 309)

Declared In
NSDictionary.h

dictionaryWithObjectsAndKeys:

Creates and returns a dictionary containing entries constructed from the specified set of values and keys.

```
+ (id)dictionaryWithObjectsAndKeys:(id)firstObject , ...
```

Parameters

firstObject
The first value to add to the new dictionary.

...
First the key for *firstObject*, then a null-terminated list of alternating values and keys. If any key is nil, an NSInvalidArgumentException is raised.

Discussion
This method is similar to dictionaryWithObjects:forKeys: (page 307), differing only in the way key-value pairs are specified.

For example:

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"value1", @"key1", @"value2", @"key2", nil];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithObjectsAndKeys: (page 324)
+ dictionaryWithObject:forKey: (page 307)
+ dictionaryWithObjects:forKeys: (page 307)
+ dictionaryWithObjects:forKeys:count: (page 308)

Declared In
NSDictionary.h

Instance Methods

allKeys

Returns a new array containing the receiver's keys.

```
- (NSArray *)allKeys
```

Return Value
A new array containing the receiver's keys, or an empty array if the receiver has no entries.

Discussion
The order of the elements in the array is not defined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allValues` (page 310)
- `allKeysForObject:` (page 310)
- `getObjects:andKeys:` (page 320)

Declared In
`NSDictionary.h`

allKeysForObject:

Returns a new array containing the keys corresponding to all occurrences of a given object in the receiver.

```
- (NSArray *)allKeysForObject:(id)anObject
```

Parameters
anObject
The value to look for in the receiver.

Return Value
A new array containing the keys corresponding to all occurrences of *anObject* in the receiver. If no object matching *anObject* is found, returns an empty array.

Discussion
Each object in the receiver is sent an `isEqual:` (page 1306) message to determine if it's equal to *anObject*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allKeys` (page 309)
- `keyEnumerator` (page 325)

Declared In
`NSDictionary.h`

allValues

Returns a new array containing the receiver's values.

```
- (NSArray *)allValues
```

Return Value
A new array containing the receiver's values, or an empty array if the receiver has no entries.

Discussion
The order of the values in the array isn't defined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allKeys` (page 309)
- `getObjects:andKeys:` (page 320)
- `objectEnumerator` (page 326)

Declared In
`NSDictionary.h`

count

Returns the number of entries in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of entries in the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDictionary.h`

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
If each key in the receiver is an `NSString` object, the entries are listed in ascending order by key, otherwise the order in which the entries are listed is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `descriptionWithLocale:` (page 312)
- `descriptionWithLocale:indent:` (page 312)

Declared In
`NSDictionary.h`

descriptionInStringsFileFormat

Returns a string that represents the contents of the receiver, formatted in `.strings` file format.

```
- (NSString *)descriptionInStringsFileFormat
```

Return Value
A string that represents the contents of the receiver, formatted in `.strings` file format.

Discussion
The order in which the entries are listed is undefined.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDictionary.h`

descriptionWithLocale:

Returns a string object that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters
locale

An object that specifies options used for formatting each of the receiver's keys and values; pass `nil` if you don't want them formatted.

Prior to Mac OS X v10.5, locale must be an instance of `NSDictionary`. With Mac OS X v10.5 and later, it may also be an `NSLocale` object.

Discussion
For a description of how *locale* is applied to each element in the receiver, see `descriptionWithLocale:indent:` (page 312).

If each key in the dictionary responds to `compare:`, the entries are listed in ascending order by key, otherwise the order in which the entries are listed is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `description` (page 311)
- `descriptionWithLocale:indent:` (page 312)

Declared In
`NSDictionary.h`

descriptionWithLocale:indent:

Returns a string object that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale indent:(NSUInteger)level
```

Parameters

locale

An object that specifies options used for formatting each of the receiver's keys and values; pass `nil` if you don't want them formatted.

Prior to Mac OS X v10.5, locale must be an instance of `NSDictionary`. With Mac OS X v10.5 and later, it may also be an `NSLocale` object.

level

Specifies a level of indent, to make the output more readable: set *level* to `0` to use four spaces to indent, or `1` to indent the output with a tab character

Return Value

A string object that represents the contents of the receiver, formatted as a property list.

Discussion

The returned `NSString` object contains the string representations of each of the receiver's entries. `descriptionWithLocale:indent:` obtains the string representation of a given key or value as follows:

- If the object is an `NSString` object, it is used as is.
- If the object responds to `descriptionWithLocale:indent:`, that method is invoked to obtain the object's string representation.
- If the object responds to `descriptionWithLocale:`, that method is invoked to obtain the object's string representation.
- If none of the above conditions is met, the object's string representation is obtained by invoking its `description` method.

If each key in the dictionary responds to `compare:`, the entries are listed in ascending order, by key. Otherwise, the order in which the entries are listed is undefined.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `- description` (page 311)
- `- descriptionWithLocale:` (page 312)

Declared In

`NSDictionary.h`

fileCreationDate

Returns the value for the `NSFileCreationDate` key.

```
- (NSDate *)fileCreationDate
```

Return Value

The value for the `NSFileCreationDate` key, or `nil` if the receiver doesn't have an entry for the key.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileExtensionHidden

Returns the value for the NSFileExtensionHidden key.

```
- (BOOL)fileExtensionHidden
```

Return Value
The value for the NSFileExtensionHidden key, or NO if the receiver doesn't have an entry for the key.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileGroupOwnerAccountID

Returns the value for the NSFileGroupOwnerAccountID key.

```
- (NSNumber *)fileGroupOwnerAccountID
```

Return Value
The value for the NSFileGroupOwnerAccountID key, or nil if the receiver doesn't have an entry for the key.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileGroupOwnerAccountName

Returns the value for the NSFileGroupOwnerAccountName key.

```
- (NSString *)fileGroupOwnerAccountName
```

Return Value
The value for the key NSFileGroupOwnerAccountName, or nil if the receiver doesn't have an entry for the key.

Discussion
This and the other file... methods are for use with a dictionary, such as those returned from the methods fileAttributesAtPath:traverseLink: (page 401) (NSFileManager), directoryAttributes (page 332) (NSDirectoryEnumerator), and fileAttributes (page 332) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory. This method returns the name of the corresponding file's group.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileHFSCreatorCode

Returns the value for the NSFileHFSCreatorCode key.

```
- (OSType)fileHFSCreatorCode
```

Return Value
The value for the NSFileHFSCreatorCode key, or 0 if the receiver doesn't have an entry for the key.

Discussion
See HFS File Types for details on the OSType data type.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileHFSTypeCode

Returns the value for the NSFileHFSTypeCode key.

```
- (OSType)fileHFSTypeCode
```

Return Value
The value for the NSFileHFSTypeCode key, or 0 if the receiver doesn't have an entry for the key.

Discussion
See HFS File Types for details on the OSType data type.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileIsAppendOnly

Returns the value for the NSFileAppendOnly key.

```
- (BOOL)fileIsAppendOnly
```

Return Value
The value for the NSFileAppendOnly key, or NO if the receiver doesn't have an entry for the key.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileIsImmutable

Returns the value for the NSFileImmutable key.

```
- (BOOL)fileIsImmutable
```

Return Value
The value for the NSFileImmutable key, or NO if the receiver doesn't have an entry for the key.

Discussion
This and the other file... methods are for use with a dictionary, such as those returned from the methods fileAttributesAtPath:traverseLink: (page 401) (NSFileManager), directoryAttributes (page 332) (NSDirectoryEnumerator), and fileAttributes (page 332) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileModificationDate

Returns the value for the key NSFileModificationDate.

```
- (NSDate *)fileModificationDate
```

Return Value
The value for the key NSFileModificationDate, or nil if the receiver doesn't have an entry for the key.

Discussion
This and the other file... methods are for use with a dictionary, such as those returned from the methods fileAttributesAtPath:traverseLink: (page 401) (NSFileManager), directoryAttributes (page 332) (NSDirectoryEnumerator), and fileAttributes (page 332) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory. This method returns the date that the file's data was last modified.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

fileOwnerAccountID

Returns the value for the NSFileOwnerAccountID key.

```
- (NSNumber *)fileOwnerAccountID
```

Return Value
The value for the `NSFileOwnerAccountID` key, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the account name of the file's owner.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

fileOwnerAccountName

Returns the value for the key `NSFileOwnerAccountName`.

```
- (NSString *)fileOwnerAccountName
```

Return Value
The value for the key `NSFileOwnerAccountName`, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the account name of the file's owner.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

filePosixPermissions

Returns the value for the key `NSFilePosixPermissions`.

```
- (NSUInteger)filePosixPermissions
```

Return Value
The value, as an `unsigned long`, for the key `NSFilePosixPermissions`, or `0` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's permissions.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

fileSize
Returns the value for the key `NSFileSize`.

```
- (unsigned long long)fileSize
```

Return Value
The value, as an `unsigned long long`, for the key `NSFileSize`, or `0` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary such, as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's size.

Special Considerations
If the file has a resource fork, the returned value does *not* include the size of the resource fork.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

fileSystemFileNumber
Returns the value for the key `NSFileSystemFileNumber`.

```
- (NSUInteger)fileSystemFileNumber
```

Return Value
The value, as an `unsigned long`, for the key `NSFileSystemFileNumber`, or `0` if the receiver doesn't have an entry for the key

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's inode.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

fileSystemNumber
Returns the value for the key `NSFileSystemNumber`.

```
- (NSInteger)fileSystemNumber
```

Return Value
The value, as an `unsigned long`, for the key `NSFileSystemNumber`, or `0` if the receiver doesn't have an entry for the key

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the ID of the device containing the file.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

fileType
Returns the value for the key `NSFileType`.

```
- (NSString *)fileType
```

Return Value
The value for the key `NSFileType`, or `nil` if the receiver doesn't have an entry for the key.

Discussion
This and the other `file...` methods are for use with a dictionary, such as those returned from the methods `fileAttributesAtPath:traverseLink:` (page 401) (`NSFileManager`), `directoryAttributes` (page 332) (`NSDirectoryEnumerator`), and `fileAttributes` (page 332) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's type. Possible return values are described in the "Constants" section of `NSFileManager`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

getObjects:andKeys:

Returns by reference C arrays of the keys and values in the receiver.

```
- (void)getObjects:(id *)objects andKeys:(id *)keys
```

Parameters
objects
Upon return, contains a C array of the values in the receiver.

keys
Upon return, contains a C array of the keys in the receiver.

Discussion
The elements in the returned arrays are ordered such that the first element in *objects* is the value for the first key in *keys* and so on.

Availability
Available in iPhone OS 2.0 and later.

See Also
- allKeys (page 309)
- allValues (page 310)
- objectForKey: (page 327)
- objectsForKeys:notFoundMarker: (page 327)

Declared In
NSDictionary.h

initWithContentsOfFile:

Initializes a newly allocated dictionary using the keys and values found in a file at a given path.

```
- (id)initWithContentsOfFile:(NSString *)path
```

Parameters
path
A full or relative pathname. The file identified by *path* must contain a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of NSData, NSDate, NSNumber, NSString, NSArray, or NSDictionary). For more details, see *Property List Programming Guide for Cocoa*.

Return Value
An initialized object—which might be different than the original receiver—that contains the dictionary at *path*, or nil if there is a file error or if the contents of the file are an invalid representation of a dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dictionaryWithContentsOfFile:` (page 305)

Declared In
`NSDictionary.h`

initWithContentsOfURL:

Initializes a newly allocated dictionary using the keys and values found at a given URL.

```
- (id)initWithContentsOfURL:(NSURL *)aURL
```

Parameters

aURL

An URL that identifies a resource containing a string representation of a property list whose root object is a dictionary. The dictionary must contain only property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`). For more details, see *Property List Programming Guide for Cocoa*.

Return Value
An initialized object—which might be different than the original receiver—that contains the dictionary at *aURL*, or `nil` if there is an error or if the contents of the resource are an invalid representation of a dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ dictionaryWithContentsOfURL:` (page 306)

Declared In
`NSDictionary.h`

initWithDictionary:

Initializes a newly allocated dictionary by placing in it the keys and values contained in another given dictionary.

```
- (id)initWithDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing keys and values for the new dictionary.

Return Value
An initialized object—which might be different than the original receiver—containing the keys and values found in *otherDictionary*.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dictionaryWithDictionary: (page 306)

Declared In
NSDictionary.h

initWithDictionary:copyItems:

Initializes a newly allocated dictionary using the objects contained in another given dictionary.

```
- (id)initWithDictionary:(NSDictionary *)otherDictionary copyItems:(BOOL)flag
```

Parameters

otherDictionary
A dictionary containing keys and values for the new dictionary.

flag
A flag that specifies whether values in *otherDictionary* should be copied. If YES, the members of *otherDictionary* are copied, and the copies are added to the receiver. If NO, the values of *otherDictionary* are retained by the new dictionary.

Return Value
An initialized object—which might be different than the original receiver—containing the keys and values found in *otherDictionary*.

Discussion
Note that copyWithZone: (page 1250) is used to make copies. Thus, the receiver's new member objects may be immutable, even though their counterparts in *otherDictionary* were mutable. Also, members must conform to the NSCopying protocol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithDictionary: (page 321)

Declared In
NSDictionary.h

initWithObjects:forKeys:

Initializes a newly allocated dictionary with entries constructed from the contents of the *objects* and *keys* arrays.

```
- (id)initWithObjects:(NSArray *)objects forKeys:(NSArray *)keys
```

Parameters

objects
An array containing the values for the new dictionary. Each value object receives a retain (page 1312) message before being added to the new dictionary.

keys

An array containing the keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 1250); keys must conform to the `NSCopying` protocol), and the copy is added to the new dictionary.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if the objects and keys arrays do not have the same number of elements.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:` (page 307)

`- initWithObjects:forKeys:count:` (page 323)

`- initWithObjectsAndKeys:` (page 324)

Declared In

`NSDictionary.h`

initWithObjects:forKeys:count:

Initializes a newly allocated dictionary with *count* entries.

```
- (id)initWithObjects:(id *)objects forKeys:(id *)keys count:(NSUInteger)count
```

Parameters

objects

A C array of values for the new dictionary. Each value object receives a `retain` (page 1312) message before being added to the new dictionary.

keys

A C array of keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 1250); keys must conform to the `NSCopying` protocol), and the copy is added to the new dictionary.

count

The number of elements to use from the *keys* and *objects* arrays. *count* must not exceed the number of elements in *objects* or *keys*.

Discussion

This method steps through the *objects* and *keys* arrays, creating entries in the new dictionary as it goes. An `NSInvalidArgumentException` is raised if a key or value object is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:count:` (page 308)

`- initWithObjects:forKeys:` (page 322)

`- initWithObjectsAndKeys:` (page 324)

Declared In

`NSDictionary.h`

initWithObjectsAndKeys:

Initializes a newly allocated dictionary with entries constructed from the specified set of values and keys.

```
- (id)initWithObjectsAndKeys:(id)firstObject , ...
```

Parameters

firstObject

The first value to add to the new dictionary.

...

First the key for *firstObject*, then a null-terminated list of alternating values and keys. If any key is `nil`, an `NSInvalidArgumentException` is raised.

Discussion

This method is similar to `initWithObjects:forKeys:` (page 322), differing only in the way in which the key-value pairs are specified.

For example:

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:
    @"value1" @"key1", @"value2", @"key2", nil];
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dictionaryWithObjectsAndKeys:` (page 309)

`- initWithObjects:forKeys:` (page 322)

`- initWithObjects:forKeys:count:` (page 323)

Declared In

`NSDictionary.h`

isEqualToDictionary:

Returns a Boolean value that indicates whether the contents of the receiver are equal to the contents of another given dictionary.

```
- (BOOL)isEqualToDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

The dictionary with which to compare the receiver.

Return Value

`YES` if the contents of *otherDictionary* are equal to the contents of the receiver, otherwise `NO`.

Discussion

Two dictionaries have equal contents if they each hold the same number of entries and, for a given key, the corresponding value objects in each dictionary satisfy the `isEqual:` (page 1306) test.

Availability

Available in iPhone OS 2.0 and later.

See Also
- isEqual: (page 1306) (NSObject protocol)

Declared In
NSDictionary.h

keyEnumerator

Returns an enumerator object that lets you access each key in the receiver.

```
- (NSEnumerator *)keyEnumerator
```

Return Value
An enumerator object that lets you access each key in the receiver.

Discussion
The following code fragment illustrates how you might use this method.

```
NSEnumerator *enumerator = [myDictionary keyEnumerator];
id key;

while ((key = [enumerator nextObject])) {
    /* code that uses the returned key */
}
```

If you use this method with instances of mutable subclasses of NSDictionary, your code should not modify the entries during enumeration. If you intend to modify the entries, use the allKeys (page 309) method to create a "snapshot" of the dictionary's keys. Then use this snapshot to traverse the entries, modifying them along the way.

Note that the objectEnumerator (page 326) method provides a convenient way to access each value in the dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
- allKeys (page 309)
- allKeysForObject: (page 310)
- getObjects:andKeys: (page 320)
- objectEnumerator (page 326)
- nextObject (page 341) (NSEnumerator)

Declared In
NSDictionary.h

keysSortedByValueUsingSelector:

Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.

```
- (NSArray *)keysSortedByValueUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that specifies the method to use to compare the values in the receiver.

The *comparator* method should return `NSOrderedAscending` if the receiver is smaller than the argument, `NSOrderedDescending` if the receiver is larger than the argument, and `NSOrderedSame` if they are equal.

Return Value

An array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.

Discussion

Pairs of dictionary values are compared using the comparison method specified by *comparator*; the *comparator* message is sent to one of the values and has as its single argument the other value from the dictionary.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `allKeys` (page 309)
- `sortedArrayUsingSelector:` (page 64) (`NSArray`)

Declared In

`NSDictionary.h`

objectEnumerator

Returns an enumerator object that lets you access each value in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value

An enumerator object that lets you access each value in the receiver.

Discussion

The following code fragment illustrates how you might use the method.

```
NSEnumerator *enumerator = [myDictionary objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the dictionary's values */
}
```

If you use this method with instances of mutable subclasses of `NSDictionary`, your code should not modify the entries during enumeration. If you intend to modify the entries, use the `allValues` (page 310) method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `keyEnumerator` (page 325)
- `nextObject` (page 341) (`NSEnumerator`)

Declared In
NSDictionary.h

objectForKey:

Returns the value associated with a given key.

```
- (id)objectForKey:(id)aKey
```

Parameters

aKey

The key for which to return the corresponding value.

Return Value
The value associated with *aKey*, or nil if no value is associated with *aKey*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- allKeys (page 309)
- allValues (page 310)
- getObjects:andKeys: (page 320)

Declared In
NSDictionary.h

objectsForKeys:notFoundMarker:

Returns the set of objects from the receiver that corresponds to the specified *keys* as an NSArray.

```
- (NSArray *)objectsForKeys:(NSArray *)keys notFoundMarker:(id)anObject
```

Parameters

keys

The keys for which to return corresponding values.

anObject

The marker object to place in the corresponding element of the returned array if an object isn't found in the receiver to correspond to a given key.

Discussion
The objects in the returned array and the *keys* array have a one-for-one correspondence, so that the *n*th object in the returned array corresponds to the *n*th key in *keys*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- allKeys (page 309)
- allValues (page 310)
- getObjects:andKeys: (page 320)

Declared In
NSDictionary.h

valueForKey:

Returns the value associated with a given key.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The key for which to return the corresponding value. Note that when using key-value coding, the key must be a string (see Key-Value Coding Fundamentals).

Return Value
The value associated with *key*.

Discussion
If *key* does not start with "@", invokes `objectForKey:` (page 327). If *key* does start with "@", strips the "@" and invokes `[super valueForKey:]` with the rest of the key.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setValue:forKey:` (page 609) (`NSMutableDictionary`)
- `getObjects:andKeys:` (page 320)

Declared In
NSKeyValueCoding.h

writeToFile:atomically:

Writes a property list representation of the contents of the receiver to a given path.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag
```

Parameters

path

The path at which to write the file.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1031) before invoking this method.

flag

A flag that specifies whether the file should be written atomically.

If *flag* is `YES`, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If *flag* is `NO`, the dictionary is written directly to *path*. The `YES` option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

Return Value
`YES` if the file is written successfully, otherwise `NO`.

Discussion

This method recursively validates that all the contained objects are property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`) before writing out the file, and returns `NO` if all the objects are not property list objects, since the resultant file would not be a valid property list.

If the receiver's contents are all property list objects, the file written by this method can be used to initialize a new dictionary with the class method `dictionaryWithContentsOfFile:` (page 305) or the instance method `initWithContentsOfFile:` (page 320).

For more information about property lists, see *Property List Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDictionary.h`

writeToURL:atomically:

Writes a property list representation of the contents of the receiver to a given URL.

```
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

Parameters

aURL

The URL to which to write the receiver.

flag

A flag that specifies whether the output should be written atomically.

If *flag* is `YES`, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If *flag* is `NO`, the dictionary is written directly to *aURL*. The `YES` option guarantees that *aURL*, if it exists at all, won't be corrupted even if the system should crash during writing. *flag* is ignored if *aURL* is of a type that cannot be written atomically.

Return Value

`YES` if the location is written successfully, otherwise `NO`.

Discussion

This method recursively validates that all the contained objects are property list objects (instances of `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray`, or `NSDictionary`) before writing out the file, and returns `NO` if all the objects are not property list objects, since the resultant output would not be a valid property list.

If the receiver's contents are all property list objects, the location written by this method can be used to initialize a new dictionary with the class method `dictionaryWithContentsOfURL:` (page 306) or the instance method `initWithContentsOfURL:` (page 321).

For more information about property lists, see *Property List Programming Guide for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDictionary.h`

NSDirectoryEnumerator Class Reference

Inherits from:	NSEnumerator : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSFileManager.h
Companion guide:	Low-Level File Management Programming Topics

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSDirectoryEnumerator` object enumerates the contents of a directory, returning the pathnames of all files and directories contained within that directory. These pathnames are relative to the directory.

An enumeration is recursive, including the files of all subdirectories, and crosses device boundaries. An enumeration does not resolve symbolic links, or attempt to traverse symbolic links that point to directories.

Tasks

Getting File and Directory Attributes

- `directoryAttributes` (page 332)
  Returns an `NSDictionary` object that contains the attributes of the directory at which enumeration started.
- `fileAttributes` (page 332)
  Returns an `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

Skipping Subdirectories

- `skipDescendents` (page 333)
  Causes the receiver to skip recursion into the most recently obtained subdirectory.

Instance Methods

directoryAttributes

Returns an `NSDictionary` object that contains the attributes of the directory at which enumeration started.

```
- (NSDictionary *)directoryAttributes
```

Return Value

An `NSDictionary` object that contains the attributes of the directory at which enumeration started.

Discussion

See the description of the `fileAttributesAtPath:traverseLink:` (page 401) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

Availability

Available in iPhone OS 2.0 and later.

See Also

`createDirectoryAtPath:attributes:` (page 394) (`NSFileManager`)

Declared In

`NSFileManager.h`

fileAttributes

Returns an `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

```
- (NSDictionary *)fileAttributes
```

Return Value
An `NSDictionary` object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

Discussion
See the description of the `fileAttributesAtPath:traverseLink:` (page 401) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

skipDescendents
Causes the receiver to skip recursion into the most recently obtained subdirectory.

```
- (void)skipDescendents
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

NSDistributedNotificationCenter Class Reference

Inherits from:	NSNotificationCenter : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Declared in:	Foundation/NSDistributedNotificationCenter.h
Companion guide:	Notification Programming Topics for Cocoa

Class at a Glance

The `NSDistributedNotificationCenter` class provides a way to send notifications to objects in other tasks. It takes `NSNotification` objects and broadcasts them to any objects in other tasks that have registered for the notification with their task's default distributed notification center.

Principal Attributes

- **Notification dispatch table.** See "Class at a Glance" > "Principal Attributes" in *NSNotificationCenter Class Reference* for information about the dispatch table.

  In addition to the notification name and sender, dispatch table entries for distributed notification centers specify when the notification center delivers notifications to its observers. See the `postNotificationName:object:userInfo:deliverImmediately:` method, "Suspending and Resuming Notification Delivery" (page ?), and `NSNotificationSuspensionBehavior` for details.

Commonly Used Methods

`defaultCenter`
: Accesses the default distributed notification center.

`addObserver:selector:name:object:suspensionBehavior:`
: Registers an object to receive a notification with a specified behavior when notification delivery is suspended.

`postNotificationName:object:userInfo:deliverImmediately:`
: Creates and posts a notification.

`removeObserver:name:object:`
: Specifies that an object no longer wants to receive certain notifications.

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSDistributedNotificationCenter` class implements a notification center that can distribute notifications asynchronously to tasks other than the one in which the notification was posted. An instance of this class are known as a **distributed notification center**.

Each task has a default distributed notification center that you access with the `defaultCenter` class method. There may be different types of distributed notification centers. Currently there is a single type—`NSLocalNotificationCenterType`. This type of distributed notification center handles notifications that can be sent between tasks on a single computer. For communication between tasks on different computers, use *Distributed Objects*.

Posting a *distributed notification* is an expensive operation. The notification gets sent to a system-wide server that distributes it to all the tasks that have objects registered for distributed notifications. The latency between posting the notification and the notification's arrival in another task is unbounded. In fact, when too many notifications are posted and the server's queue fills up, notifications may be dropped.

Distributed notifications are delivered via a task's run loop. A task must be running a run loop in one of the "common" modes, such as `NSDefaultRunLoopMode`, to receive a distributed notification. For multithreaded applications running in Mac OS X v10.3 and later, distributed notifications are always delivered to the main thread. For multithreaded applications running in Mac OS X v10.2.8 and earlier, notifications are delivered to the thread that first used the distributed notifications API, which in most cases is the main thread.

Note: `NSDistributedNotificationCenter` objects should not be used to send notifications between threads within the same task. Use *Distributed Objects* or the `NSObject` method `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811), instead. You can also setup an `NSPort` object to receive and distribute messages from other threads.

Constants

Notification Posting Behavior

These constants specify the behavior of notifications posted using the `postNotificationName:object:userInfo:options:` method.

```
enum {
    NSNotificationDeliverImmediately = (1 << 0),
    NSNotificationPostToAllSessions = (1 << 1)
};
```

Constants

`NSNotificationDeliverImmediately`
When set, the notification is delivered immediately to all observers, regardless of their suspension behavior or suspension state. When not set, allows the normal suspension behavior of notification observers to take place.

`NSNotificationPostToAllSessions`
When set, the notification is posted to all sessions. When not set, the notification is sent only to applications within the same login session as the posting task.

Declared In

`NSDistributedNotificationCenter.h`

NSEnumerator Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject) NSFastEnumeration
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSEnumerator.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSEnumerator` is an abstract class, instances of whose subclasses enumerate collections of other objects, such as arrays and dictionaries.

All creation methods are defined in the collection classes—such as `NSArray`, `NSSet`, and `NSDictionary`—which provide special `NSEnumerator` objects with which to enumerate their contents. For example, `NSArray` has two methods that return an `NSEnumerator` object: `objectEnumerator` (page 931) and `reverseObjectEnumerator` (page 60). `NSDictionary` also has two methods that return an `NSEnumerator` object: `keyEnumerator` (page 325) and `objectEnumerator` (page 326). These methods let you enumerate the contents of a dictionary by key or by value, respectively.

You send `nextObject` (page 341) repeatedly to a newly created `NSEnumerator` object to have it return the next object in the original collection. When the collection is exhausted, `nil` is returned. You cannot "reset" an enumerator after it has exhausted its collection. To enumerate a collection again, you need a new enumerator.

The enumerator subclasses used by `NSArray`, `NSDictionary`, and `NSSet` retain the collection during enumeration. When the enumeration is exhausted, the collection is released.

Note: It is not safe to modify a mutable collection while enumerating through it. Some enumerators may currently allow enumeration of a collection that is modified, but this behavior is not guaranteed to be supported in the future.

Tasks

Getting the Enumerated Objects

- `allObjects` (page 340)
  Returns an array of objects the receiver has yet to enumerate.
- `nextObject` (page 341)
  Returns the next object from the collection being enumerated.

Instance Methods

allObjects

Returns an array of objects the receiver has yet to enumerate.

```
- (NSArray *)allObjects
```

Return Value
An array of objects the receiver has yet to enumerate.

Discussion
Put another way, the array returned by this method does not contain objects that have already been enumerated with previous `nextObject` (page 341) messages.

Invoking this method exhausts the enumerator's collection so that subsequent invocations of `nextObject` return `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSEnumerator.h`

nextObject

Returns the next object from the collection being enumerated.

```
- (id)nextObject
```

Return Value

The next object from the collection being enumerated, or `nil` when all objects have been enumerated.

Discussion

The following code illustrates how this method works using an array:

```
NSArray *anArray = // ... ;
NSEnumerator *enumerator = [anArray objectEnumerator];
id object;

while ((object = [enumerator nextObject])) {
    // do something with object...
}
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSEnumerator.h`

NSError Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSError.h Foundation/NSURLError.h
Companion guide:	Error Handling Programming Guide For Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSError` object encapsulates richer and more extensible error information than is possible using only an error code or error string. The core attributes of an `NSError` object are an error domain (represented by a string), a domain-specific error code and a user info dictionary containing application specific information.

Several well-known domains are defined corresponding to Mach, POSIX, and `OSStatus` errors. In addition, `NSError` allows you to attach an arbitrary user info dictionary to an error object, and provides the means to return a human-readable description for the error.

NSError is not an abstract class, and can be used directly. Applications may choose to create subclasses of NSError to provide better localized error strings by overriding localizedDescription (page 347).

In general, a method should signal an error condition by—for example—returning NO or nil rather than by the simple presence of an error object. The method can then optionally return an NSError object by reference, in order to further describe the error.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

Tasks

Creating Error Objects

+ errorWithDomain:code:userInfo: (page 345)
Creates and initializes an NSError object for a given domain and code with a given userInfo dictionary.

- initWithDomain:code:userInfo: (page 346)
Returns an NSError object initialized for a given domain and code with a given userInfo dictionary.

Getting Error Properties

- code (page 346)
Returns the receiver's error code.

- domain (page 346)
Returns the receiver's error domain.

- userInfo (page 349)
Returns the receiver's user info dictionary.

Getting a Localized Error Description

- localizedDescription (page 347)
Returns a string containing the localized description of the error.

- `localizedRecoveryOptions` (page 348)
  Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.
- `localizedRecoverySuggestion` (page 348)
  Returns a string containing the localized recovery suggestion for the error.
- `localizedFailureReason` (page 348)
  Returns a string containing the localized explanation of the reason for the error.

Getting the Error Recovery Attempter

- `recoveryAttempter` (page 349)
  Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

Class Methods

errorWithDomain:code:userInfo:

Creates and initializes an `NSError` object for a given domain and code with a given `userInfo` dictionary.

```
+ (id)errorWithDomain:(NSString *)domain code:(NSInteger)code userInfo:(NSDictionary
    *)dict
```

Parameters

domain
  The error domain—this can be one of the predefined `NSError` domains, or an arbitrary string describing a custom domain. *domain* must not be `nil`.

code
  The error code for the error.

dict
  The `userInfo` dictionary for the error. *userInfo* may be `nil`.

Return Value

An `NSError` object for *domain* with the specified error *code* and the dictionary of arbitrary data *userInfo*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSError.h`

Instance Methods

code

Returns the receiver's error code.

```
- (NSInteger)code
```

Return Value
The receiver's error code.

Discussion
Note that errors are domain specific.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `localizedDescription` (page 347)
- `domain` (page 346)
- `userInfo` (page 349)

Declared In
`NSError.h`

domain

Returns the receiver's error domain.

```
- (NSString *)domain
```

Return Value
A string containing the receiver's error domain.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `code` (page 346)
- `localizedDescription` (page 347)
- `userInfo` (page 349)

Declared In
`NSError.h`

initWithDomain:code:userInfo:

Returns an `NSError` object initialized for a given domain and code with a given `userInfo` dictionary.

```
- (id)initWithDomain:(NSString *)domain code:(NSInteger)code userInfo:(NSDictionary
    *)dict
```

Parameters

domain

The error domain—this can be one of the predefined `NSError` domains, or an arbitrary string describing a custom domain. *domain* must not be `nil`.

code

The error code for the error.

dict

The `userInfo` dictionary for the error. *userInfo* may be `nil`.

Return Value

An `NSError` object initialized for *domain* with the specified error *code* and the dictionary of arbitrary data *userInfo*.

Discussion

This is the designated initializer for `NSError`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ errorWithDomain:code:userInfo:` (page 345)

Declared In

`NSError.h`

localizedDescription

Returns a string containing the localized description of the error.

```
- (NSString *)localizedDescription
```

Return Value

A string containing the localized description of the error.

By default this method returns the object in the user info dictionary for the key `NSLocalizedDescriptionKey`. If the user info dictionary doesn't contain a value for `NSLocalizedDescriptionKey`, a default string is constructed from the domain and code.

Discussion

This method can be overridden by subclasses to present customized error strings.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `code` (page 346)
- `domain` (page 346)
- `userInfo` (page 349)

Declared In

`NSError.h`

localizedFailureReason

Returns a string containing the localized explanation of the reason for the error.

```
- (NSString *)localizedFailureReason
```

Return Value

A string containing the localized explanation of the reason for the error. By default this method returns the object in the user info dictionary for the key `NSLocalizedFailureReasonErrorKey`.

Discussion

This method can be overridden by subclasses to present customized error strings.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `code` (page 346)
- `domain` (page 346)
- `userInfo` (page 349)

Declared In

`NSError.h`

localizedRecoveryOptions

Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

```
- (NSArray *)localizedRecoveryOptions
```

Return Value

An array containing the localized titles of buttons appropriate for displaying in an alert panel. By default this method returns the object in the user info dictionary for the key `NSLocalizedRecoveryOptionsErrorKey`. If the user info dictionary doesn't contain a value for `NSLocalizedRecoveryOptionsErrorKey`, `nil` is returned.

Discussion

The first string is the title of the right-most and default button, the second the one to the left of that, and so on. The recovery options should be appropriate for the recovery suggestion returned by `localizedRecoverySuggestion` (page 348). If the user info dictionary doesn't contain a value for `NSLocalizedRecoveryOptionsErrorKey`, only an OK button is displayed.

This method can be overridden by subclasses to present customized recovery suggestion strings.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSError.h`

localizedRecoverySuggestion

Returns a string containing the localized recovery suggestion for the error.

```
- (NSString *)localizedRecoverySuggestion
```

Return Value
A string containing the localized recovery suggestion for the error. By default this method returns the object in the user info dictionary for the key `NSLocalizedRecoverySuggestionErrorKey`. If the user info dictionary doesn't contain a value for `NSLocalizedRecoverySuggestionErrorKey`, `nil` is returned.

Discussion
The returned string is suitable for displaying as the secondary message in an alert panel.

This method can be overridden by subclasses to present customized recovery suggestion strings.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSError.h`

recoveryAttempter

Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

```
- (id)recoveryAttempter
```

Return Value
An object that conforms to the `NSErrorRecoveryAttempting` informal protocol. By default this method returns the object for the user info dictionary for the key `NSRecoveryAttempterErrorKey`. If the user info dictionary doesn't contain a value for `NSRecoveryAttempterErrorKey`, `nil` is returned.

Discussion
The recovery attempter must be an object that can correctly interpret an index into the array returned by `localizedRecoveryOptions` (page 348).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `localizedRecoveryOptions` (page 348)

Declared In
`NSError.h`

userInfo

Returns the receiver's user info dictionary.

```
- (NSDictionary *)userInfo
```

Return Value
The receiver's user info dictionary, or `nil` if the user info dictionary has not been set.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `code` (page 346)
- `domain` (page 346)
- `localizedDescription` (page 347)

Declared In
`NSError.h`

Constants

User info dictionary keys

These keys may exist in the user info dictionary.

```
extern NSString *NSLocalizedDescriptionKey;
extern NSString *NSErrorFailingURLStringKey;
const NSString *NSFilePathErrorKey;
const NSString *NSStringEncodingErrorKey;
const NSString *NSUnderlyingErrorKey;
const NSString *NSURLErrorKey;
const NSString *NSLocalizedFailureReasonErrorKey;
const NSString *NSLocalizedRecoverySuggestionErrorKey;
const NSString *NSLocalizedRecoveryOptionsErrorKey;
const NSString *NSRecoveryAttempterErrorKey;
```

Constants

`NSLocalizedDescriptionKey`

The corresponding value is a localized string representation of the error that, if present, will be returned by `localizedDescription` (page 347).

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSErrorFailingURLStringKey`

The corresponding value is the URL that caused the error. This key is only present in the `NSURLErrorDomain`.

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSFilePathErrorKey`

Contains the file path of the error.

The corresponding value is an `NSString` object.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSStringEncodingErrorKey`

The corresponding value is an `NSNumber` object containing the `NSStringEncoding` value.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSUnderlyingErrorKey`

The corresponding value is an error that was encountered in an underlying implementation and caused the error that the receiver represents to occur.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSURLErrorKey`

The corresponding value is an `NSURL` object.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSLocalizedFailureReasonErrorKey`

The corresponding value is a localized string representation containing the reason for the failure that, if present, will be returned by `localizedFailureReason` (page 348).

This string provides a more detailed explanation of the error than the description.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSLocalizedRecoverySuggestionErrorKey`

The corresponding value is a string containing the localized recovery suggestion for the error.

This string is suitable for displaying as the secondary message in an alert panel.

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSLocalizedRecoveryOptionsErrorKey`

The corresponding value is an array containing the localized titles of buttons appropriate for displaying in an alert panel.

The first string is the title of the right-most and default button, the second the one to the left, and so on. The recovery options should be appropriate for the recovery suggestion returned by `localizedRecoverySuggestion` (page 348).

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSRecoveryAttempterErrorKey`

The corresponding value is an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

The recovery attempter must be an object that can correctly interpret an index into the array returned by `recoveryAttempter` (page 349).

Available in Mac OS X 10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

Declared In
`NSError.h`

Error Domains

The following error domains are predefined.

```
const NSString *NSPOSIXErrorDomain;
const NSString *NSOSStatusErrorDomain;
const NSString *NSMachErrorDomain;
```

Constants

`NSPOSIXErrorDomain`

POSIX/BSD errors

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSOSStatusErrorDomain`

Mac OS 9/Carbon errors

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

`NSMachErrorDomain`

Mach errors

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

Discussion
Additionally, the following error domain is defined by Core Foundation:

`CFStreamErrorDomain`	Defines constants for values returned in the domain field of the CFStreamError structure.

Declared In
`NSError.h`

NSException Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSException.h
Companion guide:	Exception Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSException` is used to implement exception handling and contains information about an exception. An exception is a special condition that interrupts the normal flow of program execution. Each application can interrupt the program for different reasons. For example, one application might interpret saving a file in a directory that is write-protected as an exception. In this sense, the exception is equivalent to an error. Another application might interpret the user's keypress (for example, Control-C) as an exception: an indication that a long-running process should be aborted.

Note: The exception handling mechanism uses `longjmp` to control the flow of execution. Any code written for an application that uses exception handling is therefore subject to the restrictions associated with this functionality. See your compiler documentation for more information on the `longjmp` function.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

Tasks

Creating and Raising an NSException Object

`+ exceptionWithName:reason:userInfo:` (page 355)
Creates and returns an exception object .

`+ raise:format:` (page 355)
A convenience method that creates and raises an exception.

`+ raise:format:arguments:` (page 356)
Creates and raises an exception with the specified name, reason, and arguments.

`- initWithName:reason:userInfo:` (page 357)
Initializes and returns a newly allocated exception object.

`- raise` (page 358)
Raises the receiver, causing program flow to jump to the local exception handler.

Querying an NSException Object

`- name` (page 358)
Returns an `NSString` object used to uniquely identify the receiver.

`- reason` (page 358)
Returns an `NSString` object containing a “human-readable” reason for the receiver.

`- userInfo` (page 359)
Returns an `NSDictionary` object containing application-specific data pertaining to the receiver.

Getting Exception Stack Frames

- callStackReturnAddresses (page 357)
  Returns the call return addresses related to a raised exception.

Class Methods

exceptionWithName:reason:userInfo:

Creates and returns an exception object .

```
+ (NSException *)exceptionWithName:(NSString *)name reason:(NSString *)reason
    userInfo:(NSDictionary *)userInfo
```

Parameters

name
The name of the exception.

reason
A human-readable message string summarizing the reason for the exception.

userInfo
A dictionary containing user-defined information relating to the exception

Return Value

The created NSException object or nil if the object couldn't be created.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithName:reason:userInfo: (page 357)
- name (page 358)
- reason (page 358)
- userInfo (page 359)

Declared In

NSException.h

raise:format:

A convenience method that creates and raises an exception.

```
+ (void)raise:(NSString *)name format:(NSString *)format, ...
```

Parameters

name
The name of the exception.

format,
A human-readable message string (that is, the exception reason) with conversion specifications for the variable arguments that follow.

`...`
: Variable information to be inserted into the formatted exception reason (in the manner of `printf`).

Discussion
The user-defined information is `nil` for the generated exception object.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ raise:format:arguments:` (page 356)
`- raise` (page 358)

Declared In
`NSException.h`

raise:format:arguments:

Creates and raises an exception with the specified name, reason, and arguments.

```
+ (void)raise:(NSString *)name format:(NSString *)format arguments:(va_list)argList
```

Parameters

name
: The name of the exception.

format
: A human-readable message string (that is, the exception reason) with conversion specifications for the variable arguments in `argList`.

argList
: Variable information to be inserted into the formatted exception reason (in the manner of `vprintf`).

Discussion
The user-defined dictionary of the generated object is `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ raise:format:` (page 355)
`- raise` (page 358)

Declared In
`NSException.h`

Instance Methods

callStackReturnAddresses

Returns the call return addresses related to a raised exception.

```
- (NSArray *)callStackReturnAddresses
```

Return Value

An array of `NSNumber` objects encapsulating `NSUInteger` values. Each value is a call frame return address. The array of stack frames starts at the point at which the exception was first raised, with the first items being the most recent stack frames.

Discussion

`NSException` subclasses posing as the `NSException` class or subclasses or other API elements that interfere with the exception-raising mechanism may not get this information.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSException.h`

initWithName:reason:userInfo:

Initializes and returns a newly allocated exception object.

```
- (id)initWithName:(NSString *)name reason:(NSString *)reason userInfo:(NSDictionary
    *)userInfo
```

Parameters

name

The name of the exception.

reason

A human-readable message string summarizing the reason for the exception.

userInfo

A dictionary containing user-defined information relating to the exception

Return Value

The created `NSException` object or `nil` if the object couldn't be created.

Discussion

This is the designated initializer.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ exceptionWithName:reason:userInfo:` (page 355)

Declared In

`NSException.h`

name

Returns an `NSString` object used to uniquely identify the receiver.

```
- (NSString *)name
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 355)
`- initWithName:reason:userInfo:` (page 357)

Declared In
`NSException.h`

raise

Raises the receiver, causing program flow to jump to the local exception handler.

```
- (void)raise
```

Discussion
All other methods that raise an exception invoke this method, so set a breakpoint here if you are debugging exceptions. When there are no exception handlers in the exception handler stack, unless the exception is raised during the posting of a notification, this method calls the uncaught exception handler, in which last-minute logging can be performed. The program then terminates, regardless of the actions taken by the uncaught exception handler.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ raise:format:` (page 355)
`+ raise:format:arguments:` (page 356)

Declared In
`NSException.h`

reason

Returns an `NSString` object containing a “human-readable” reason for the receiver.

```
- (NSString *)reason
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 355)
`- initWithName:reason:userInfo:` (page 357)

Declared In
NSException.h

userInfo

Returns an NSDictionary object containing application-specific data pertaining to the receiver.

```
- (NSDictionary *)userInfo
```

Discussion
Returns nil if no application-specific data exists. As an example, if a method's return value caused the exception to be raised, the return value might be available to the exception handler through this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ exceptionWithName:reason:userInfo: (page 355)
- initWithName:reason:userInfo: (page 357)

Declared In
NSException.h

Constants

The string constants for exceptions are listed and described in the "Constants" (page 1413) chapter.

NSFileHandle Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSFileHandle.h
Companion guide:	Low-Level File Management Programming Topics

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSFileHandle` objects provide an object-oriented wrapper for accessing open files or communications channels.

Tasks

Getting a File Handle

+ `fileHandleForReadingAtPath:` (page 364)
  Returns a file handle initialized for reading the file, device, or named socket at the specified path.

+ `fileHandleForWritingAtPath:` (page 365)
  Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

+ `fileHandleForUpdatingAtPath:` (page 364)
  Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.

+ `fileHandleWithStandardError` (page 366)
  Returns the file handle associated with the standard error file.

+ `fileHandleWithStandardInput` (page 366)
  Returns the file handle associated with the standard input file.

+ `fileHandleWithStandardOutput` (page 367)
  Returns the file handle associated with the standard output file.

+ `fileHandleWithNullDevice` (page 365)
  Returns a file handle associated with a null device.

Creating a File Handle

- `initWithFileDescriptor:` (page 370)
  Returns a file handle initialized with a file descriptor.

- `initWithFileDescriptor:closeOnDealloc:` (page 370)
  Returns a file handle initialized with a file handle, using a specified deallocation policy.

Getting a File Descriptor

- `fileDescriptor` (page 369)
  Returns the file descriptor associated with the receiver.

Reading from a File Handle

- `availableData` (page 368)
  Returns the data available through the receiver.

- `readDataToEndOfFile` (page 372)
  Returns the data available through the receiver up to the end of file or maximum number of bytes.

- `readDataOfLength:` (page 371)
  Reads data up to a specified number of bytes from the receiver.

Writing to a File Handle

- `writeData:` (page 377)
  Synchronously writes data to the file, device, pipe, or socket represented by the receiver.

Communicating Asynchronously

- `acceptConnectionInBackgroundAndNotify` (page 367)
  Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the “near” (client) end of the communications channel.
- `acceptConnectionInBackgroundAndNotifyForModes:` (page 368)
  Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the “near” (client) end of the communications channel.
- `readInBackgroundAndNotify` (page 372)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readInBackgroundAndNotifyForModes:` (page 373)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotify` (page 373)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 374)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `waitForDataInBackgroundAndNotify` (page 376)
  Checks to see if data is available in a background thread.
- `waitForDataInBackgroundAndNotifyForModes:` (page 376)
  Checks to see if data is available in a background thread.

Seeking Within a File

- `offsetInFile` (page 371)
  Returns the position of the file pointer within the file represented by the receiver.
- `seekToEndOfFile` (page 375)
  Puts the file pointer at the end of the file referenced by the receiver and returns the new file offset.
- `seekToFileOffset:` (page 375)
  Moves the file pointer to the specified offset within the file represented by the receiver.

Operating on a File

- `closeFile` (page 369)
  Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.

- synchronizeFile (page 375)
  Causes all in-memory data and attributes of the file represented by the receiver to be written to permanent storage.
- truncateFileAtOffset: (page 376)
  Truncates or extends the file represented by the receiver to a specified offset within the file and puts the file pointer at that position.

Class Methods

fileHandleForReadingAtPath:

Returns a file handle initialized for reading the file, device, or named socket at the specified path.

```
+ (id)fileHandleForReadingAtPath:(NSString *)path
```

Parameters

path
  The path to the file, device, or named socket to access.

Return Value

The initialized file handle, or `nil` if no file exists at *path*.

Discussion

The file pointer is set to the beginning of the file. The returned object responds only to `NSFileHandle read...` messages.

Availability

Available in iPhone OS 2.0 and later.

See Also

- availableData (page 368)
- initWithFileDescriptor: (page 370)
- readDataOfLength: (page 371)
- readDataToEndOfFile (page 372)

Declared In

NSFileHandle.h

fileHandleForUpdatingAtPath:

Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.

```
+ (id)fileHandleForUpdatingAtPath:(NSString *)path
```

Parameters

path
  The path to the file, device, or named socket to access.

Return Value

The initialized file handle, or `nil` if no file exists at *path*.

Discussion
The file pointer is set to the beginning of the file. The returned object responds to both `NSFileHandle read...` messages and `writeData:` (page 377).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `availableData` (page 368)
- `initWithFileDescriptor:` (page 370)
- `readDataOfLength:` (page 371)
- `readDataToEndOfFile` (page 372)

Declared In
`NSFileHandle.h`

fileHandleForWritingAtPath:
Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

```
+ (id)fileHandleForWritingAtPath:(NSString *)path
```

Parameters
path

The path to the file, device, or named socket to access.

Return Value
The initialized file handle, or `nil` if no file exists at *path*.

Discussion
The file pointer is set to the beginning of the file. The returned object responds only to `writeData:` (page 377).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithFileDescriptor:` (page 370)

Declared In
`NSFileHandle.h`

fileHandleWithNullDevice
Returns a file handle associated with a null device.

```
+ (id)fileHandleWithNullDevice
```

Return Value
A file handle associated with a null device.

Discussion
You can use null-device file handles as "placeholders" for standard-device file handles or in collection objects to avoid exceptions and other errors resulting from messages being sent to invalid file handles. Read messages sent to a null-device file handle return an end-of-file indicator (an empty `NSData` object) rather than raise an exception. Write messages are no-ops, whereas `fileDescriptor` (page 369) returns an illegal value. Other methods are no-ops or return "sensible" values.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithFileDescriptor:` (page 370)

Declared In
`NSFileHandle.h`

fileHandleWithStandardError

Returns the file handle associated with the standard error file.

```
+ (id)fileHandleWithStandardError
```

Return Value
The shared file handle associated with the standard error file.

Discussion
Conventionally this is a terminal device to which error messages are sent. There is one standard error file handle per process; it is a shared instance.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ fileHandleWithNullDevice` (page 365)
`- initWithFileDescriptor:` (page 370)

Declared In
`NSFileHandle.h`

fileHandleWithStandardInput

Returns the file handle associated with the standard input file.

```
+ (id)fileHandleWithStandardInput
```

Return Value
The shared file handle associated with the standard input file.

Discussion
Conventionally this is a terminal device on which the user enters a stream of data. There is one standard input file handle per process; it is a shared instance.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ fileHandleWithNullDevice (page 365)
- initWithFileDescriptor: (page 370)

Declared In
NSFileHandle.h

fileHandleWithStandardOutput

Returns the file handle associated with the standard output file.

```
+ (id)fileHandleWithStandardOutput
```

Return Value
The shared file handle associated with the standard output file.

Discussion
Conventionally this is a terminal device that receives a stream of data from a program. There is one standard output file handle per process; it is a shared instance.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ fileHandleWithNullDevice (page 365)
- initWithFileDescriptor: (page 370)

Declared In
NSFileHandle.h

Instance Methods

acceptConnectionInBackgroundAndNotify

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.

```
- (void)acceptConnectionInBackgroundAndNotify
```

Discussion
This method is asynchronous. In a separate "safe" thread it accepts a connection, creates a file handle for the other end of the connection, and returns that object to the client by posting an NSFileHandleConnectionAcceptedNotification (page 379) in the run loop of the client. The notification includes as data a *userInfo* dictionary containing the created NSFileHandle object; access this object using the NSFileHandleNotificationFileHandleItem key.

The receiver must be created by an initWithFileDescriptor: (page 370) message that takes as an argument a stream-type socket created by the appropriate system routine. The object that will write data to the returned file handle must add itself as an observer of NSFileHandleConnectionAcceptedNotification (page 379).

Note that this method does not continue to listen for connection requests after it posts `NSFileHandleConnectionAcceptedNotification`. If you want to keep getting notified, you need to call `acceptConnectionInBackgroundAndNotify` again in your observer method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)
`- readInBackgroundAndNotify` (page 372)
`- readToEndOfFileInBackgroundAndNotify` (page 373)

Declared In
`NSFileHandle.h`

acceptConnectionInBackgroundAndNotifyForModes:

Accepts a socket connection (for stream-type sockets only) in the background and creates a file handle for the "near" (client) end of the communications channel.

```
- (void)acceptConnectionInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters
modes
The runloop modes in which the connection accepted notification can be posted.

Discussion
See `acceptConnectionInBackgroundAndNotify` (page 367) for details of how this method operates. This method differs from `acceptConnectionInBackgroundAndNotify` (page 367) in that *modes* specifies the run-loop mode (or modes) in which `NSFileHandleConnectionAcceptedNotification` (page 379) can be posted.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)
`- readInBackgroundAndNotifyForModes:` (page 373)
`- readToEndOfFileInBackgroundAndNotifyForModes:` (page 374)

Declared In
`NSFileHandle.h`

availableData

Returns the data available through the receiver.

```
- (NSData *)availableData
```

Return Value
The data currently available through the receiver.

Discussion
If the receiver is a file, returns the data obtained by reading the file from the file pointer to the end of the file. If the receiver is a communications channel, reads up to a buffer of data and returns it; if no data is available, the method blocks. Returns an empty data object if the end of file is reached. Raises `NSFileHandleOperationException` if attempts to determine file-handle type fail or if attempts to read from the file or channel fail.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `readDataOfLength:` (page 371)
- `readDataToEndOfFile` (page 372)

Declared In
`NSFileHandle.h`

closeFile
Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.

```
- (void)closeFile
```

Discussion
The file or communications channel is available for other uses after the file handle is closed. Further read and write messages sent to a file handle to which `closeFile` has been sent raises an exception.

Sending `closeFile` to a file handle does not cause its deallocation. Deallocation of a file handle deletes its descriptor and closes the represented file or channel only if the handle was created using `initWithFileDescriptor:closeOnDealloc:` (page 370) with `YES` as the *`flag`* argument.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

fileDescriptor
Returns the file descriptor associated with the receiver.

```
- (int)fileDescriptor
```

Return Value
The POSIX file descriptor associated with the receiver.

Discussion
You can send this message to file handles originating from both file descriptors and file handles and receive a valid file descriptor so long as the file handle is open. If the file handle has been closed by sending it `closeFile` (page 369), this method raises an exception.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithFileDescriptor:` (page 370)

Declared In
`NSFileHandle.h`

initWithFileDescriptor:

Returns a file handle initialized with a file descriptor.

```
- (id)initWithFileDescriptor:(int)fileDescriptor
```

Parameters

fileDescriptor
The POSIX file descriptor with which to initialize the file handle.

Return Value
A file handle initialized with *fileDescriptor*.

Discussion
You can create a file handle for a socket by using the result of a `socket` call as *fileDescriptor*.

Special Considerations

The object creating a file handle using this method owns *fileDescriptor* and is responsible for its disposition.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `closeFile` (page 369)

Declared In
`NSFileHandle.h`

initWithFileDescriptor:closeOnDealloc:

Returns a file handle initialized with a file handle, using a specified deallocation policy.

```
- (id)initWithFileDescriptor:(int)fileDescriptor closeOnDealloc:(BOOL)flag
```

Parameters

fileDescriptor
The POSIX file descriptor with which to initialize the file handle.

flag
`YES` if the file descriptor should be closed when the receiver is deallocated, otherwise `NO`.

Return Value
A file handle initialized with *fileDescriptor* with a deallocation policy specified by *flag*.

Special Considerations

If *flag* is NO, the object creating a file handle using this method owns *fileDescriptor* and is responsible for its disposition.

Availability

Available in iPhone OS 2.0 and later.

See Also

- closeFile (page 369)

Declared In

NSFileHandle.h

offsetInFile

Returns the position of the file pointer within the file represented by the receiver.

```
- (unsigned long long)offsetInFile
```

Return Value

The position of the file pointer within the file represented by the receiver.

Special Considerations

Raises an exception if the message is sent to a file handle representing a pipe or socket or if the file descriptor is closed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- seekToEndOfFile (page 375)
- seekToFileOffset: (page 375)

Declared In

NSFileHandle.h

readDataOfLength:

Reads data up to a specified number of bytes from the receiver.

```
- (NSData *)readDataOfLength:(NSUInteger)length
```

Parameters

length

The number of bytes to read from the receiver.

Return Value

The data available through the receiver up to a maximum of *length* bytes.

Discussion

If the receiver is a file, returns the data obtained by reading from the file pointer to *length* or to the end of the file, whichever comes first. If the receiver is a communications channel, the method reads data from the channel up to *length*. Returns an empty NSData object if the file is positioned at the

end of the file or if an end-of-file indicator is returned on a communications channel. Raises `NSFileHandleOperationException` if attempts to determine file-handle type fail or if attempts to read from the file or channel fail.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `availableData` (page 368)
- `readDataToEndOfFile` (page 372)

Declared In
`NSFileHandle.h`

readDataToEndOfFile

Returns the data available through the receiver up to the end of file or maximum number of bytes.

```
- (NSData *)readDataToEndOfFile
```

Return Value
The data available through the receiver up to `UINT_MAX` bytes (the maximum value for unsigned integers) or, if a communications channel, until an end-of-file indicator is returned.

Discussion
This method invokes `readDataOfLength:` (page 371) as part of its implementation.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `availableData` (page 368)

Declared In
`NSFileHandle.h`

readInBackgroundAndNotify

Reads from the file or communications channel in the background and posts a notification when finished.

```
- (void)readInBackgroundAndNotify
```

Discussion
This method performs an asynchronous `availableData` (page 368) operation on a file or communications channel and posts an `NSFileHandleReadCompletionNotification` (page 380) to the client process's run loop.

The length of the data is limited to the buffer size of the underlying operating system. The notification includes a *`userInfo`* dictionary that contains the data read; access this object using the `NSFileHandleNotificationDataItem` key.

Any object interested in receiving this data asynchronously must add itself as an observer of `NSFileHandleReadCompletionNotification` (page 380). In communication via stream-type sockets, the receiver is often the object returned in the *`userInfo`* dictionary of `NSFileHandleConnectionAcceptedNotification` (page 379).

Note that this method does not cause a continuous stream of notifications to be sent. If you wish to keep getting notified, you'll also need to call `readInBackgroundAndNotify` in your observer method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `acceptConnectionInBackgroundAndNotify` (page 367)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 374)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)

Declared In
`NSFileHandle.h`

readInBackgroundAndNotifyForModes:

Reads from the file or communications channel in the background and posts a notification when finished.

```
- (void)readInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters
`modes`

The runloop modes in which the read completion notification can be posted.

Discussion
See `readInBackgroundAndNotify` (page 372) for details of how this method operates. This method differs from `readInBackgroundAndNotify` (page 372) in that *`modes`* specifies the run-loop mode (or modes) in which `NSFileHandleReadCompletionNotification` (page 380) can be posted.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `acceptConnectionInBackgroundAndNotifyForModes:` (page 368)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)

Declared In
`NSFileHandle.h`

readToEndOfFileInBackgroundAndNotify

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

```
- (void)readToEndOfFileInBackgroundAndNotify
```

Discussion

This method performs an asynchronous `readToEndOfFile` operation on a file or communications channel and posts an `NSFileHandleReadToEndOfFileCompletionNotification` (page 380) to the client process's run loop.

The notification includes a *`userInfo`* dictionary that contains the data read; access this object using the `NSFileHandleNotificationDataItem` key.

Any object interested in receiving this data asynchronously must add itself as an observer of `NSFileHandleReadToEndOfFileCompletionNotification` (page 380). In communication via stream-type sockets, the receiver is often the object returned in the *`userInfo`* dictionary of `NSFileHandleConnectionAcceptedNotification` (page 379).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `acceptConnectionInBackgroundAndNotify` (page 367)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 374)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)

Declared In

`NSFileHandle.h`

readToEndOfFileInBackgroundAndNotifyForModes:

Reads to the end of file from the file or communications channel in the background and posts a notification when finished.

```
- (void)readToEndOfFileInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters

`modes`

The runloop modes in which the read completion notification can be posted.

Discussion

See `readToEndOfFileInBackgroundAndNotify` (page 373) for details of this method's operation. The method differs from `readToEndOfFileInBackgroundAndNotify` (page 373) in that *`modes`* specifies the run-loop mode (or modes) in which `NSFileHandleReadToEndOfFileCompletionNotification` (page 380) can be posted.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `acceptConnectionInBackgroundAndNotifyForModes:` (page 368)
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690) (`NSNotificationQueue`)

Declared In

`NSFileHandle.h`

seekToEndOfFile

Puts the file pointer at the end of the file referenced by the receiver and returns the new file offset.

```
- (unsigned long long)seekToEndOfFile
```

Return Value

The file offset with the file pointer at the end of the file. This is therefore equal to the size of the file.

Special Considerations

Raises an exception if the message is sent to an `NSFileHandle` object representing a pipe or socket or if the file descriptor is closed.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- offsetInFile` (page 371)

Declared In

`NSFileHandle.h`

seekToFileOffset:

Moves the file pointer to the specified offset within the file represented by the receiver.

```
- (void)seekToFileOffset:(unsigned long long)offset
```

Parameters

offset

The offset to seek to.

Special Considerations

Raises an exception if the message is sent to an `NSFileHandle` object representing a pipe or socket, if the file descriptor is closed, or if any other error occurs in seeking.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- offsetInFile` (page 371)

Declared In

`NSFileHandle.h`

synchronizeFile

Causes all in-memory data and attributes of the file represented by the receiver to be written to permanent storage.

```
- (void)synchronizeFile
```

Discussion
This method should be invoked by programs that require the file to always be in a known state. An invocation of this method does not return until memory is flushed.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileHandle.h

truncateFileAtOffset:

Truncates or extends the file represented by the receiver to a specified offset within the file and puts the file pointer at that position.

```
- (void)truncateFileAtOffset:(unsigned long long)offset
```

Parameters
offset
The offset within the file that will mark the new end of the file.

Discussion
If the file is extended (if *offset* is beyond the current end of file), the added characters are null bytes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileHandle.h

waitForDataInBackgroundAndNotify

Checks to see if data is available in a background thread.

```
- (void)waitForDataInBackgroundAndNotify
```

Discussion
When the data becomes available, the thread notifies all observers with NSFileHandleDataAvailableNotification (page 379). After the notification has been posted, the thread is terminated.

Availability
Available in iPhone OS 2.0 and later.

See Also
- waitForDataInBackgroundAndNotifyForModes: (page 376)

Declared In
NSFileHandle.h

waitForDataInBackgroundAndNotifyForModes:

Checks to see if data is available in a background thread.

```
- (void)waitForDataInBackgroundAndNotifyForModes:(NSArray *)modes
```

Parameters

modes

The runloop modes in which the data available notification can be posted.

Discussion

When the data becomes available, the thread notifies all observers with `NSFileHandleDataAvailableNotification` (page 379). After the notification has been posted, the thread is terminated. This method differs from `waitForDataInBackgroundAndNotify` (page 376) in that *modes* specifies the run-loop mode (or modes) in which `NSFileHandleDataAvailableNotification` (page 379) can be posted.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `waitForDataInBackgroundAndNotify` (page 376)

Declared In

`NSFileHandle.h`

writeData:

Synchronously writes data to the file, device, pipe, or socket represented by the receiver.

```
- (void)writeData:(NSData *)data
```

Parameters

data

The data to be written.

Discussion

If the receiver is a file, writing takes place at the file pointer's current position. After it writes the data, the method advances the file pointer by the number of bytes written. Raises an exception if the file descriptor is closed or is not valid, if the receiver represents an unconnected pipe or socket endpoint, if no free space is left on the file system, or if any other writing error occurs.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `availableData` (page 368)
- `readDataOfLength:` (page 371)
- `readDataToEndOfFile` (page 372)

Declared In

`NSFileHandle.h`

Constants

Keys for Notification UserInfo Dictionary

Strings that are used as keys in a userinfo dictionary in a file handle notification.

```
NSString * const NSFileHandleNotificationFileHandleItem;
NSString * const NSFileHandleNotificationDataItem;
```

Constants

`NSFileHandleNotificationFileHandleItem`

A key in the userinfo dictionary in a NSFileHandleConnectionAcceptedNotification (page 379) notification.

The corresponding value is the `NSFileHandle` object representing the "near" end of a socket connection.

Available in iPhone OS 2.0 and later.

Declared in `NSFileHandle.h`

`NSFileHandleNotificationDataItem`

A key in the userinfo dictionary in a NSFileHandleReadCompletionNotification (page 380) and NSFileHandleReadToEndOfFileCompletionNotification (page 380).

The corresponding value is an `NSData` object containing the available data read from a socket connection.

Available in iPhone OS 2.0 and later.

Declared in `NSFileHandle.h`

Declared In

`NSFileHandle.h`

Exception Names

Constant that defines the name of a file operation exception.

```
extern NSString *NSFileHandleOperationException;
```

Constants

`NSFileHandleOperationException`

Raised by `NSFileHandle` if attempts to determine file-handle type fail or if attempts to read from a file or channel fail.

Available in iPhone OS 2.0 and later.

Declared in `NSFileHandle.h`

Declared In

`NSFileHandle.h`

Unused Constant

Constant that is currently unused.

```
NSString * const NSFileHandleNotificationMonitorModes;
```

Constants

`NSFileHandleNotificationMonitorModes`

Currently unused.

Available in iPhone OS 2.0 and later.

Declared in `NSFileHandle.h`

Declared In

`NSFileHandle.h`

Notifications

`NSFileHandle` posts several notifications related to asynchronous background I/O operations. They are set to post when the run loop of the thread that started the asynchronous operation is idle.

NSFileHandleConnectionAcceptedNotification

This notification is posted when an `NSFileHandle` object establishes a socket connection between two processes, creates an `NSFileHandle` object for one end of the connection, and makes this object available to observers by putting it in the *`userInfo`* dictionary. To cause the posting of this notification, you must send either `acceptConnectionInBackgroundAndNotify` (page 367) or `acceptConnectionInBackgroundAndNotifyForModes:` (page 368) to an `NSFileHandle` object representing a server stream-type socket.

The notification object is the `NSFileHandle` object that sent the notification. The *`userInfo`* dictionary contains the following information:

Key	Value
`NSFileHandleNotificationFileHandleItem`	The `NSFileHandle` object representing the "near" end of a socket connection
`@"NSFileHandleError"`	An `NSNumber` object containing an integer representing the UNIX-type error which occurred

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSFileHandle.h`

NSFileHandleDataAvailableNotification

This notification is posted when the background thread determines that data is currently available for reading in a file or at a communications channel. The observers can then issue the appropriate messages to begin reading the data. To cause the posting of this notification, you must send either `waitForDataInBackgroundAndNotify` (page 376) or `waitForDataInBackgroundAndNotifyForModes:` (page 376) to an appropriate `NSFileHandle` object.

The notification object is the `NSFileHandle` object that sent the notification. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

NSFileHandleReadCompletionNotification

This notification is posted when the background thread reads the data currently available in a file or at a communications channel. It makes the data available to observers by putting it in the *`userInfo`* dictionary. To cause the posting of this notification, you must send either `readInBackgroundAndNotify` (page 372) or `readInBackgroundAndNotifyForModes:` (page 373) to an appropriate `NSFileHandle` object.

The notification object is the `NSFileHandle` object that sent the notification. The *`userInfo`* dictionary contains the following information:

Key	Value
`NSFileHandleNotificationDataItem`	An `NSData` object containing the available data read from a socket connection
`@"NSFileHandleError"`	An `NSNumber` object containing an integer representing the UNIX-type error which occurred

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

NSFileHandleReadToEndOfFileCompletionNotification

This notification is posted when the background thread reads all data in the file or, if a communications channel, until the other process signals the end of data. It makes the data available to observers by putting it in the *`userInfo`* dictionary. To cause the posting of this notification, you must send either `readToEndOfFileInBackgroundAndNotify` (page 373) or `readToEndOfFileInBackgroundAndNotifyForModes:` (page 374) to an appropriate `NSFileHandle` object.

The notification object is the `NSFileHandle` object that sent the notification. The *`userInfo`* dictionary contains the following information:

Key	Value
`NSFileHandleNotificationDataItem`	An `NSData` object containing the available data read from a socket connection
`@"NSFileHandleError"`	An `NSNumber` object containing an integer representing the UNIX-type error which occurred

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

NSFileManager Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSFileManager.h
Companion guide:	Low-Level File Management Programming Topics

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSFileManager` enables you to perform many generic file-system operations and insulates an application from the underlying file system.

Tasks

Getting the Default Manager

`+ defaultManager` (page 388)
Returns the default `NSFileManager` object for the file system.

Moving an Item

`- fileManager:shouldMoveItemAtPath:toPath:` (page 414) *delegate method*
An `NSFileManager` object sends this message immediately before attempting to move to a given path.

`- moveItemAtPath:toPath:error:` (page 408)
Moves the directory or file specified by a given path to a different location in the file system identified by another path.

`- fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 417) *delegate method*
An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given path.

Copying an Item

`- fileManager:shouldCopyItemAtPath:toPath:` (page 413) *delegate method*
An `NSFileManager` object sends this message immediately before attempting to copy to a given path.

`- copyItemAtPath:toPath:error:` (page 393)
Copies the directory or file specified in a given path to a different location in the file system identified by another path.

`- fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 416) *delegate method*
An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.

Removing an Item

`- fileManager:shouldRemoveItemAtPath:` (page 418) *delegate method*
An `NSFileManager` object sends this message immediately before attempting to copy to a given path.

`- removeItemAtPath:error:` (page 409)
Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

`- fileManager:shouldProceedAfterError:removingItemAtPath:` (page 417) *delegate method*
An `NSFileManager` object sends this message if an error occurs during an attempt to delete a given path.

Creating an Item

- `createDirectoryAtPath:attributes:` (page 394)
  Creates a directory (without contents) at a given path with given attributes.
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 394)
  Creates a directory with given attributes at a specified path.
- `createFileAtPath:contents:attributes:` (page 395)
  Creates a file at a given path that has given attributes and contents.

Linking an Item

- `fileManager:shouldLinkItemAtPath:toPath:` (page 413) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to link to a given path.
- `linkItemAtPath:toPath:error:` (page 407)
  Creates a link from a source to a destination.
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 416) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to link to a given path.

Symbolic-Link Operations

- `createSymbolicLinkAtPath:pathContent:` (page 396)
  Creates a symbolic link identified by a given path that refers to a given location.
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 396)
  Creates a symbolic link identified by a given path that refers to a given location.
- `pathContentOfSymbolicLinkAtPath:` (page 408)
  Returns the path of the directory or file that a symbolic link at a given path refers to.
- `destinationOfSymbolicLinkAtPath:error:` (page 398)
  Returns an `NSString` object containing the path of the item pointed at by the symlink specified by a given path.

Handling File Operations

The methods described in this section are methods to be implemented by the callback handler passed to several methods of `NSFileManager`.

- `fileManager:shouldProceedAfterError:` (page 414) *delegate method*
  An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.
- `fileManager:willProcessPath:` (page 419) *delegate method*
  An `NSFileManager` object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

Getting and Comparing File Contents

- `contentsAtPath:` (page 391)
  Returns as an `NSData` object the contents of the file at at given path.
- `contentsEqualAtPath:andPath:` (page 392)
  Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

Discovering Directory Contents

- `directoryContentsAtPath:` (page 398)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `contentsOfDirectoryAtPath:error:` (page 392)
  Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.
- `enumeratorAtPath:` (page 400)
  Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.
- `subpathsAtPath:` (page 411)
  Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.
- `subpathsOfDirectoryAtPath:error:` (page 412)
  Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

Determining Access to Files

- `fileExistsAtPath:` (page 402)
  Returns a Boolean value that indicates whether a file exists at a given path.
- `fileExistsAtPath:isDirectory:` (page 403)
  Returns a Boolean value that indicates whether a specified file exists.
- `isReadableFileAtPath:` (page 406)
  Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.
- `isWritableFileAtPath:` (page 406)
  Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.
- `isExecutableFileAtPath:` (page 405)
  Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.
- `isDeletableFileAtPath:` (page 405)
  Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

Getting and Setting Attributes

- `componentsToDisplayForPath:` (page 391)
  Returns an array of `NSString` objects representing the user-visible components of a given path.
- `displayNameAtPath:` (page 399)
  Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.
- `fileAttributesAtPath:traverseLink:` (page 401)
  Returns a dictionary that describes the POSIX attributes of the file specified at a given.
- `attributesOfItemAtPath:error:` (page 389)
  An `NSDictionary` object containing the attributes of the item at a given path.
- `fileSystemAttributesAtPath:` (page 403)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `attributesOfFileSystemForPath:error:` (page 388)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `changeFileAttributes:atPath:` (page 390)
  Changes the attributes of a given file or directory.
- `setAttributes:ofItemAtPath:error:` (page 409)
  Sets the attributes of a given file or directory.

Getting Representations of File Paths

- `fileSystemRepresentationWithPath:` (page 404)
  Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.
- `stringWithFileSystemRepresentation:length:` (page 411)
  Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

Managing the Delegate

- `setDelegate:` (page 410)
  Sets the delegate for the receiver.
- `delegate` (page 398)
  Returns the delegate for the receiver.

Managing the Current Directory

- `changeCurrentDirectoryPath:` (page 389)
  Changes the path of the current directory for the current process to a given path.
- `currentDirectoryPath` (page 397)
  Returns the path of the program's current directory.

Class Methods

defaultManager

Returns the default `NSFileManager` object for the file system.

```
+ (NSFileManager *)defaultManager
```

Return Value
The default `NSFileManager` object for the file system.

Discussion
You invoke all `NSFileManager` instance methods with this object as the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

Instance Methods

attributesOfFileSystemForPath:error:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)attributesOfFileSystemForPath:(NSString *)path error:(NSError
    **)error
```

Parameters
path
: Any pathname within the mounted file system.

error
: If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSDictionary` object that describes the attributes of the mounted file system on which *path* resides. See "File-System Attribute Keys" (page 424) for a description of the keys available in the dictionary.

Discussion
This method does not traverse an initial symbolic link.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `fileSystemAttributesAtPath:` (page 403)

- fileAttributesAtPath:traverseLink: (page 401)
- changeFileAttributes:atPath: (page 390)

Declared In
NSFileManager.h

attributesOfItemAtPath:error:

An NSDictionary object containing the attributes of the item at a given path.

```
- (NSDictionary *)attributesOfItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters
path

The path of a file or directory.

error

If an error occurs, upon return contains an NSError object that describes the problem. Pass NULL if you do not want error information.

Return Value
An NSDictionary object that describes the attributes (file, directory, symlink, and so on) of the file specified by *path*. The keys in the dictionary are described in "File Attribute Keys" (page 419).

Discussion
This method does not traverse an initial symbolic link.

Availability
Available in iPhone OS 2.0 and later.

See Also
- fileAttributesAtPath:traverseLink: (page 401)
- changeFileAttributes:atPath: (page 390)

Declared In
NSFileManager.h

changeCurrentDirectoryPath:

Changes the path of the current directory for the current process to a given path.

```
- (BOOL)changeCurrentDirectoryPath:(NSString *)path
```

Parameters
path

The path of the directory to which to change.

Return Value
YES if successful, otherwise NO.

Discussion
All relative pathnames refer implicitly to the current working directory. The current working directory is stored per process.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `currentDirectoryPath` (page 397)
- `fileExistsAtPath:isDirectory:` (page 403)
- `directoryContentsAtPath:` (page 398)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 394)
- `createDirectoryAtPath:attributes:` (page 394)

Declared In
`NSFileManager.h`

changeFileAttributes:atPath:
Changes the attributes of a given file or directory.

```
- (BOOL)changeFileAttributes:(NSDictionary *)attributes atPath:(NSString *)path
```

Parameters
attributes

A dictionary containing as keys the attributes to set for *path* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

For the `NSFilePosixPermissions` value, specify a file mode from the OR'd permission bit masks defined in `sys/stat.h`. See the man page for the `chmod` function (`man 2 chmod`) for an explanation.

path

A path to a file or directory.

Return Value
`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion
As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *path* specifies a file.

Special Considerations
On Mac OS X v10.5 and later, use `setAttributes:ofItemAtPath:error:` (page 409) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `fileAttributesAtPath:traverseLink:` (page 401)
- `setAttributes:ofItemAtPath:error:` (page 409)

Declared In
`NSFileManager.h`

componentsToDisplayForPath:

Returns an array of `NSString` objects representing the user-visible components of a given path.

```
- (NSArray *)componentsToDisplayForPath:(NSString *)path
```

Parameters

path

A pathname.

Return Value

An array of `NSString` objects representing the user-visible (for the Finder, Open and Save panels, and so on) components of *path*.

Discussion

These components cannot be used for path operations and are only suitable for display to the user.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSFileManager.h`

contentsAtPath:

Returns as an `NSData` object the contents of the file at at given path.

```
- (NSData *)contentsAtPath:(NSString *)path
```

Parameters

path

The path of a file.

Return Value

The contents of the file specified by *path* as an `NSData` object. If *path* specifies a directory, or if some other error occurs, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `contentsEqualAtPath:andPath:` (page 392)
- `createFileAtPath:contents:attributes:` (page 395)

Declared In

`NSFileManager.h`

contentsEqualAtPath:andPath:

Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

```
- (BOOL)contentsEqualAtPath:(NSString *)path1 andPath:(NSString *)path2
```

Parameters

path1

The path of a file or directory to compare with the contents of *path2*.

path2

The path of a file or directory to compare with the contents of *path1*.

Return Value

`YES` if file or directory specified in *path1* has the same contents as that specified in *path2*, otherwise `NO`.

Discussion

If *path1* and *path2* are directories, the contents are the list of files and subdirectories each contains—contents of subdirectories are also compared. For files, this method checks to see if they're the same file, then compares their size, and finally compares their contents. This method does not traverse symbolic links, but compares the links themselves.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- contentsAtPath:` (page 391)

Declared In

`NSFileManager.h`

contentsOfDirectoryAtPath:error:

Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)contentsOfDirectoryAtPath:(NSString *)patherror:(NSError **)error
```

Parameters

path

A path to a directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

An array of `NSString` objects identifying the directories and files (including symbolic links) contained in *path*. Returns an empty array if the directory exists but has no contents. Returns `nil` if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion
The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory (“.”), parent directory (“..”), or resource forks (begin with “._”) and does not traverse symbolic links.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `directoryContentsAtPath:` (page 398)
- `currentDirectoryPath` (page 397)
- `fileExistsAtPath:isDirectory:` (page 403)
- `enumeratorAtPath:` (page 400)
- `subpathsAtPath:` (page 411)

Declared In
`NSFileManager.h`

copyItemAtPath:toPath:error:
Copies the directory or file specified in a given path to a different location in the file system identified by another path.

```
- (BOOL)copyItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError **)error
```

Parameters
srcPath
The path of a file or directory.

dstPath
The path of a file or directory.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `fileManager:shouldCopyItemAtPath:toPath:` (page 413)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 416)
- `linkItemAtPath:toPath:error:` (page 407)
- `moveItemAtPath:toPath:error:` (page 408)
- `removeItemAtPath:error:` (page 409)

Declared In
`NSFileManager.h`

createDirectoryAtPath:attributes:

Creates a directory (without contents) at a given path with given attributes.

```
- (BOOL)createDirectoryAtPath:(NSString *)path attributes:(NSDictionary *)attributes
```

Parameters

path

The path at which to create the new directory. The directory to be created must not yet exist, but its parent directory must exist.

attributes

The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, default values for these attributes are set (particularly write access for the creator and read access for others). The "Constants" (page 419) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Special Considerations

On Mac OS X v10.5 and later, use `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 394) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 394)
- `changeCurrentDirectoryPath:` (page 389)
- `changeFileAttributes:atPath:` (page 390)
- `createFileAtPath:contents:attributes:` (page 395)
- `currentDirectoryPath` (page 397)

Declared In

`NSFileManager.h`

createDirectoryAtPath:withIntermediateDirectories:attributes:error:

Creates a directory with given attributes at a specified path.

```
- (BOOL)createDirectoryAtPath:(NSString
    *)pathwithIntermediateDirectories:(BOOL)createIntermediatesattributes:(NSDictionary
    *)attributeserror:(NSError **)error
```

Parameters

path

The path at which to create the new directory. The directory to be created must not yet exist.

createIntermediates

If `YES`, then the method will also create any necessary intermediate directories; if `NO`, then the method will fail if any parent of the directory to be created does not exist.

attributes
The file attributes for the new directory. The attributes you can set are owner and group numbers, file permissions, and modification date. If you specify `nil` for *attributes*, the directory is created according to the umask of the process. The "Constants" (page 419) section lists the global constants used as keys in the *attributes* dictionary. Some of the keys, such as `NSFileHFSCreatorCode` and `NSFileHFSTypeCode`, do not apply to directories.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `createDirectoryAtPath:attributes:` (page 394)
- `changeCurrentDirectoryPath:` (page 389)
- `setAttributes:ofItemAtPath:error:` (page 409)
- `createFileAtPath:contents:attributes:` (page 395)
- `currentDirectoryPath` (page 397)

Declared In
`NSFileManager.h`

createFileAtPath:contents:attributes:

Creates a file at a given path that has given attributes and contents.

```
- (BOOL)createFileAtPath:(NSString *)path contents:(NSData *)contents
    attributes:(NSDictionary *)attributes
```

Parameters

path
The path for the new file.

contents
The contents for the new file.

attributes
A dictionary that describes the attributes of the new file. The file attributes you can set are owner and group numbers, file permissions, and modification date. "File Attribute Keys" (page 419) lists the global constants used as keys in the *attributes* dictionary. If you specify `nil` for *attributes*, the file is given a default set of attributes.

Return Value
`YES` if the operation was successful, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `contentsAtPath:` (page 391)

- changeFileAttributes:atPath: (page 390)
- setAttributes:ofItemAtPath:error: (page 409)
- fileAttributesAtPath:traverseLink: (page 401)
- attributesOfItemAtPath:error: (page 389)

Declared In
NSFileManager.h

createSymbolicLinkAtPath:pathContent:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)path pathContent:(NSString *)otherPath
```

Parameters
path
The path for a symbolic link.

otherPath
The path to which *path* should refer.

Return Value
YES if the operation is successful, otherwise NO. Returns NO if a file, directory, or symbolic link identical to *path* already exists.

Discussion
Creates a symbolic link identified by *path* that refers to the location *otherPath* in the file system.

Special Considerations
On Mac OS X v10.5 and later, use createSymbolicLinkAtPath:withDestinationPath:error: (page 396) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- createSymbolicLinkAtPath:withDestinationPath:error: (page 396)
- pathContentOfSymbolicLinkAtPath: (page 408)

Declared In
NSFileManager.h

createSymbolicLinkAtPath:withDestinationPath:error:

Creates a symbolic link identified by a given path that refers to a given location.

```
- (BOOL)createSymbolicLinkAtPath:(NSString *)pathwithDestinationPath:(NSString
    *)destPatherror:(NSError **)error
```

Parameters
path
The path for a symbolic link.

destPath
The path to which *path* should refer.

error
If an error occurs, upon return contains an NSError object that describes the problem. Pass NULL if you do not want error information.

Return Value
YES if the operation is successful, otherwise NO. Returns NO if a file, directory, or symbolic link identical to *path* already exists.

Discussion
Creates a symbolic link identified by *path* that refers to the location *destPath* in the file system.

This method does not traverse an initial symlink.

Availability
Available in iPhone OS 2.0 and later.

See Also
- createSymbolicLinkAtPath:pathContent: (page 396)
- pathContentOfSymbolicLinkAtPath: (page 408)

Declared In
NSFileManager.h

currentDirectoryPath

Returns the path of the program's current directory.

```
- (NSString *)currentDirectoryPath
```

Return Value
The path of the program's current directory. If the program's current working directory isn't accessible, returns nil.

Discussion
The string returned by this method is initialized to the current working directory; you can change the working directory by invoking changeCurrentDirectoryPath: (page 389).

Relative pathnames refer implicitly to the current directory. For example, if the current directory is /tmp, and the relative pathname reports/info.txt is specified, the resulting full pathname is /tmp/reports/info.txt.

Availability
Available in iPhone OS 2.0 and later.

See Also
- changeCurrentDirectoryPath: (page 389)
- createDirectoryAtPath:attributes: (page 394)
- createDirectoryAtPath:withIntermediateDirectories:attributes:error: (page 394)

Declared In
NSFileManager.h

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

destinationOfSymbolicLinkAtPath:error:

Returns an `NSString` object containing the path of the item pointed at by the symlink specified by a given path.

```
- (NSString *)destinationOfSymbolicLinkAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path
The path of a file or directory.

error
If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value
An `NSString` object containing the path of the directory or file to which the symbolic link *path* refers, or `nil` upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Discussion
This method does not traverse an initial symlink.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `pathContentOfSymbolicLinkAtPath:` (page 408)
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 396)

Declared In
NSFileManager.h

directoryContentsAtPath:

Returns an array of `NSString` objects identifying the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)directoryContentsAtPath:(NSString *)path
```

Parameters

path

A path to a directory.

Return Value

An array of `NSString` objects identifying the directories and files (including symbolic links) contained in *path*. Returns an empty array if the directory exists but has no contents. Returns `nil` if the directory specified at *path* does not exist or there is some other error accessing it.

Discussion

The search is shallow and therefore does not return the contents of any subdirectories. This returned array does not contain strings for the current directory (“.”), parent directory (“..”), or resource forks (begin with “._”) and does not traverse symbolic links.

Special Considerations

On Mac OS X v10.5 and later, use `contentsOfDirectoryAtPath:error:` (page 392) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `contentsOfDirectoryAtPath:error:` (page 392)
- `currentDirectoryPath` (page 397)
- `fileExistsAtPath:isDirectory:` (page 403)
- `enumeratorAtPath:` (page 400)
- `subpathsAtPath:` (page 411)

Declared In

`NSFileManager.h`

displayNameAtPath:

Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.

```
- (NSString *)displayNameAtPath:(NSString *)path
```

Parameters

path

The path of a file or directory.

Return Value

The name of the file or directory at *path* in a localized form appropriate for presentation to the user. If there is no file or directory at *path*, or if an error occurs, returns `[path lastPathComponent]`.

Discussion

The returned value is localized where appropriate. For example, if you have selected French as your preferred language, the following code fragment logs “Bibliothèque”:

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory,
NSUserDomainMask, YES);
if ([paths count] > 0)
{
    NSString *documentsDirectory = [paths objectAtIndex:0];
```

```
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *displayNameAtPath = [fileManager
displayNameAtPath:documentsDirectory];
    NSLog(@"%@", displayNameAtPath);
}
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `lastPathComponent` (page 1011) (`NSString`)

Declared In
`NSFileManager.h`

enumeratorAtPath:

Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.

```
- (NSDirectoryEnumerator *)enumeratorAtPath:(NSString *)path
```

Parameters
path

The path of the directory to enumerate.

Return Value
An `NSDirectoryEnumerator` object that enumerates the contents of the directory at *path*. If *path* is a symbolic link, this method evaluates the link and returns an enumerator for the file or directory the link points to. If the link cannot be evaluated, the method returns `nil`.

If *path* is a filename, the method returns an enumerator object that enumerates no files—the first call to `nextObject` (page 341) will return `nil`.

Discussion
Because the enumeration is deep—that is, it lists the contents of all subdirectories—this enumerator object is useful for performing actions that involve large file-system subtrees. If the method is passed a directory on which another file system is mounted (a mount point), it traverses the mount point. This method does not resolve symbolic links encountered in the traversal process, nor does it recurse through them if they point to a directory.

This code fragment enumerates the subdirectories and files under a user's `Documents` directory and processes all files with an extension of `.doc`:

```
NSString *file;
NSString *docsDir = [NSHomeDirectory() stringByAppendingPathComponent:
@"Documents"];
NSDirectoryEnumerator *dirEnum =
    [[NSFileManager defaultManager] enumeratorAtPath:docsDir];

while (file = [dirEnum nextObject]) {
    if ([[file pathExtension] isEqualToString: @"doc"]) {
        [self scanDocument: [docsDir stringByAppendingPathComponent:file]];
    }
}
```

The NSDirectoryEnumerator class has methods for obtaining the attributes of the existing path and of the parent directory and for skipping descendants of the existing path.

Availability
Available in iPhone OS 2.0 and later.

See Also
- currentDirectoryPath (page 397)
- fileAttributesAtPath:traverseLink: (page 401)
- directoryContentsAtPath: (page 398)
- subpathsAtPath: (page 411)

Declared In
NSFileManager.h

fileAttributesAtPath:traverseLink:
Returns a dictionary that describes the POSIX attributes of the file specified at a given.

```
- (NSDictionary *)fileAttributesAtPath:(NSString *)path traverseLink:(BOOL)flag
```

Parameters
path

A file path.

flag

If *path* is not a symbolic link, this parameter has no effect. If *path* is a symbolic link, then:

- If YES the attributes of the linked-to file are returned, or if the link points to a nonexistent file the method returns nil.
- If NO, the attributes of the symbolic link are returned.

Return Value
An NSDictionary object that describes the POSIX attributes of the file specified at *path*. The keys in the dictionary are described in "File Attribute Keys" (page 419). If there is no item at *path*, returns nil.

Discussion
This code example gets several attributes of a file and logs them.

```
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = @"/tmp/List";
NSDictionary *fileAttributes = [fileManager fileAttributesAtPath:path
traverseLink:YES];

if (fileAttributes != nil) {
    NSNumber *fileSize;
    NSString *fileOwner;
    NSDate *fileModDate;
    if (fileSize = [fileAttributes objectForKey:NSFileSize]) {
        NSLog(@"File size: %qi\n", [fileSize unsignedLongLongValue]);
    }
    if (fileOwner = [fileAttributes objectForKey:NSFileOwnerAccountName]) {
        NSLog(@"Owner: %@\n", fileOwner);
    }
```

```
    if (fileModDate = [fileAttributes objectForKey:NSFileModificationDate]) {
        NSLog(@"Modification date: %@\n", fileModDate);
    }
}
else {
    NSLog(@"Path (%@) is invalid.", path);
}
```

As a convenience, `NSDictionary` provides a set of methods (declared as a category in `NSFileManager.h`) for quickly and efficiently obtaining attribute information from the returned dictionary: `fileGroupOwnerAccountName` (page 314), `fileModificationDate` (page 316), `fileOwnerAccountName` (page 317), `filePosixPermissions` (page 317), `fileSize` (page 318), `fileSystemFileNumber` (page 318), `fileSystemNumber` (page 319), and `fileType` (page 319). For example, you could rewrite the file modification statement in the code example above as:

```
if (fileModDate = [fileAttributes fileModificationDate])
    NSLog(@"Modification date: %@\n", fileModDate);
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfItemAtPath:error:` (page 389) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `attributesOfItemAtPath:error:` (page 389)
- `changeFileAttributes:atPath:` (page 390)

Declared In

`NSFileManager.h`

fileExistsAtPath:

Returns a Boolean value that indicates whether a file exists at a given path.

```
- (BOOL)fileExistsAtPath:(NSString *)path
```

Parameters

path

The path of a file or directory. If *path* begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath` (page 1031), or this method will return `NO`.

Return Value

`YES` if a file specified in *path* exists, otherwise `NO`. If *path* specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file at the link destination.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `fileExistsAtPath:isDirectory:` (page 403)

Declared In

`NSFileManager.h`

fileExistsAtPath:isDirectory:

Returns a Boolean value that indicates whether a specified file exists.

```
- (BOOL)fileExistsAtPath:(NSString *)path isDirectory:(BOOL *)isDirectory
```

Parameters

path

The path of a file or directory. If *path* begins with a tilde (~), it must first be expanded with `stringByExpandingTildeInPath` (page 1031), or this method will return `NO`.

isDirectory

Upon return, contains `YES` if *path* is a directory or if the final path element is a symbolic link that points to a directory, otherwise contains `NO`. If *path* doesn't exist, the return value is undefined. Pass `NULL` if you do not need this information.

Return Value

`YES` if there is a file or directory at *path*, otherwise `NO`. If *path* specifies a symbolic link, this method traverses the link and returns `YES` or `NO` based on the existence of the file or directory at the link destination.

Discussion

If you need to further determine if *path* is a package, use the `NSWorkspace` method `isFilePackageAtPath:`.

This example gets an array that identifies the fonts in the user's fonts directory:

```
NSArray *subpaths;
BOOL isDir;

NSArray *paths = NSSearchPathForDirectoriesInDomains
                    (NSLibraryDirectory, NSUserDomainMask, YES);

if ([paths count] == 1) {

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *fontPath = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Fonts"];

    if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir) {
        subpaths = [fileManager subpathsAtPath:fontPath];
// ...
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`- fileExistsAtPath:` (page 402)

Declared In

`NSFileManager.h`

fileSystemAttributesAtPath:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.

```
- (NSDictionary *)fileSystemAttributesAtPath:(NSString *)path
```

Parameters

path

Any pathname within the mounted file system.

Return Value

An `NSDictionary` object that describes the attributes of the mounted file system on which *path* resides. See "File-System Attribute Keys" (page 424) for a description of the keys available in the dictionary.

Discussion

The following code example checks to see if there's sufficient space on the file system before adding a new file to it:

```
NSData *contents = [myImage TIFFRepresentation];
NSFileManager *fileManager = [NSFileManager defaultManager];
NSString *path = ...;
NSString *fileName = ...;
NSDictionary *fsAttributes =
        [fileManager fileSystemAttributesAtPath:path];
if ([[fsAttributes objectForKey:NSFileSystemFreeSize] unsignedLongLongValue]
>
        [contents length])
    [fileManager createFileAtPath:[path stringByAppendingPathComponent:fileName]
             contents:contents attributes:nil];
```

Special Considerations

On Mac OS X v10.5 and later, use `attributesOfFileSystemForPath:error:` (page 388) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `attributesOfFileSystemForPath:error:` (page 388)
- `fileAttributesAtPath:traverseLink:` (page 401)
- `changeFileAttributes:atPath:` (page 390)

Declared In

`NSFileManager.h`

fileSystemRepresentationWithPath:

Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.

```
- (const char *)fileSystemRepresentationWithPath:(NSString *)path
```

Parameters

path

A file path.

Return Value

A C-string representation of *path* that properly encodes Unicode strings for use by the file system.

Discussion
If you need the C string beyond the scope of your autorelease pool, you must copy it. This method raises an exception upon error. Use this method if your code calls system routines that expect C-string path arguments.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stringWithFileSystemRepresentation:length:` (page 411)

Declared In
`NSFileManager.h`

isDeletableFileAtPath:
Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

```
- (BOOL)isDeletableFileAtPath:(NSString *)path
```

Parameters
path
A file path.

Return Value
`YES` if the invoking object appears able to delete the file specified in *path*, otherwise `NO`. If the file at *path* does not exist, this method returns `NO`.

Discussion
For a directory or file to be able to be deleted, either the parent directory of *path* must be writable or its owner must be the same as the owner of the application process. If *path* is a directory, every item contained in *path* must be able to be deleted.

This method does not traverse symbolic links.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileManager.h`

isExecutableFileAtPath:
Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.

```
- (BOOL)isExecutableFileAtPath:(NSString *)path
```

Parameters
path
A file path.

Return Value
YES if the operating system appears able to execute the file specified in *path*, otherwise NO. If the file at *path* does not exist, this method returns NO.

Discussion
This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is executable.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

isReadableFileAtPath:
Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.

```
- (BOOL)isReadableFileAtPath:(NSString *)path
```

Parameters
path
A file path.

Return Value
YES if the invoking object appears able to read the file specified in *path*, otherwise NO. If the file at *path* does not exist, this method returns NO.

Discussion
This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is readable.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

isWritableFileAtPath:
Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.

```
- (BOOL)isWritableFileAtPath:(NSString *)path
```

Parameters
path
A file path.

Return Value
YES if the invoking object appears able to write to the file specified in *path*, otherwise NO. If the file at *path* does not exist, this method returns NO.

Discussion
This method traverses symbolic links. This method uses the real user ID and group ID, as opposed to the effective user and group IDs, to determine if the file is writable.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSFileManager.h

linkItemAtPath:toPath:error:
Creates a link from a source to a destination.

```
- (BOOL)linkItemAtPath:(NSString *)srcPathtoPath:(NSString *)dstPatherror:(NSError
    **)error
```

Parameters
srcPath

A path that identifies a source file or directory.

The file, link, or directory specified by *srcPath* must exist.

dstPath

A path that identifies a destination file or directory.

The destination should not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

error

If an error occurs, upon return contains an NSError object that describes the problem. Pass NULL if you do not want error information.

Return Value
YES if the link operation is successful, otherwise NO.

Discussion
If pathname *srcPath* identifies a file, this method hard-links the file specified in *dstPath* to it. If *srcPath* is a directory or symbolic link, this method copies it to *dstPath* instead of creating a hard link. Symbolic links in *srcPath* are not traversed.

Availability
Available in iPhone OS 2.0 and later.

See Also
- fileManager:shouldLinkItemAtPath:toPath: (page 413)
- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath: (page 416)
- createSymbolicLinkAtPath:withDestinationPath:error: (page 396)
- copyItemAtPath:toPath:error: (page 393)
- moveItemAtPath:toPath:error: (page 408)

Declared In
NSFileManager.h

moveItemAtPath:toPath:error:

Moves the directory or file specified by a given path to a different location in the file system identified by another path.

```
- (BOOL)moveItemAtPath:(NSString *)srcPathtoPath:(NSString *)dstPatherror:(NSError **)error
```

Parameters

srcPath

The path of a file or directory to move. *srcPath* must exist.

dstPath

The path to which the file or directory at *srcPath* is moved. *destination* must not yet exist. The destination path must end in a filename; there is no implicit adoption of the source filename.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the move operation is successful, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `fileManager:shouldMoveItemAtPath:toPath:` (page 414)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 417)

Declared In

`NSFileManager.h`

pathContentOfSymbolicLinkAtPath:

Returns the path of the directory or file that a symbolic link at a given path refers to.

```
- (NSString *)pathContentOfSymbolicLinkAtPath:(NSString *)path
```

Parameters

path

The path of a symbolic link.

Return Value

The path of the directory or file to which the symbolic link *path* refers, or `nil` upon failure. If the symbolic link is specified as a relative path, that relative path is returned.

Special Considerations

On Mac OS X v10.5 and later, use `destinationOfSymbolicLinkAtPath:error:` (page 398) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `destinationOfSymbolicLinkAtPath:error:` (page 398)
- `createSymbolicLinkAtPath:pathContent:` (page 396)

Declared In
NSFileManager.h

removeItemAtPath:error:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.

```
- (BOOL)removeItemAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

The path of a file, link, or directory to delete. The value must not be "." or "..".

error

If an error occurs, upon return contains an NSError object that describes the problem. Pass NULL if you do not want error information.

Return Value
YES if the removal operation is successful, otherwise NO.

Discussion
Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *path* an NSInvalidArgumentException exception is raised. This method does not traverse symbolic links.

Availability
Available in iPhone OS 2.0 and later.

See Also
- copyItemAtPath:toPath:error: (page 393)
- linkItemAtPath:toPath:error: (page 407)
- moveItemAtPath:toPath:error: (page 408)
- fileManager:shouldRemoveItemAtPath: (page 418)
- fileManager:shouldProceedAfterError:removingItemAtPath: (page 417)

Declared In
NSFileManager.h

setAttributes:ofItemAtPath:error:

Sets the attributes of a given file or directory.

```
- (BOOL)setAttributes:(NSDictionary *)attributes ofItemAtPath:(NSString
    *)path error:(NSError **)error
```

Parameters

attributes

A dictionary containing as keys the attributes to set for *path* and as values the corresponding value for the attribute. You can set following: `NSFileBusy`, `NSFileCreationDate`, `NSFileExtensionHidden`, `NSFileGroupOwnerAccountID`, `NSFileGroupOwnerAccountName`, `NSFileHFSCreatorCode`, `NSFileHFSTypeCode`, `NSFileImmutable`, `NSFileModificationDate`, `NSFileOwnerAccountID`, `NSFileOwnerAccountName`, `NSFilePosixPermissions`. You can change single attributes or any combination of attributes; you need not specify keys for all attributes.

path

The path of a file or directory.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if *all* changes succeed. If any change fails, returns `NO`, but it is undefined whether any changes actually occurred.

Discussion

As in the POSIX standard, the application either must own the file or directory or must be running as superuser for attribute changes to take effect. The method attempts to make all changes specified in attributes and ignores any rejection of an attempted modification.

The `NSFilePosixPermissions` value must be initialized with the code representing the POSIX file-permissions bit pattern. `NSFileHFSCreatorCode` and `NSFileHFSTypeCode` will only be heeded when *path* specifies a file.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSFileManager.h`

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate

The delegate for the receiver.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSFileManager.h`

stringWithFileSystemRepresentation:length:

Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

```
- (NSString *)stringWithFileSystemRepresentation:(const char *)string
    length:(NSUInteger)len
```

Parameters

string

A C string representation of a pathname.

len

The number of characters in *string*.

Return Value

An `NSString` object converted from the C-string representation *string* with length *len* of a pathname in the current file system.

Discussion

Use this method if your code receives paths as C strings from system routines.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- fileSystemRepresentationWithPath:` (page 404)

Declared In

`NSFileManager.h`

subpathsAtPath:

Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.

```
- (NSArray *)subpathsAtPath:(NSString *)path
```

Parameters

path

The path of the directory to list.

Return Value

An array that contains (as `NSString` objects) the contents of the directory identified by *path*. If *path* is a symbolic link, `subpathsAtPath:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion

This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips "." and "..".

This method reveals every element of the subtree at *path*, including the contents of file packages (such as applications, nib files, and RTFD files). This code fragment gets the contents of `/System/Library/Fonts` after verifying that the directory exists:

```
BOOL isDir=NO;
NSArray *subpaths;
```

```
NSString *fontPath = @"/System/Library/Fonts";
NSFileManager *fileManager = [NSFileManager defaultManager];
if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir)
    subpaths = [fileManager subpathsAtPath:fontPath];
```

Special Considerations

On Mac OS X v10.5 and later, use `subpathsOfDirectoryAtPath:error:` (page 412) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `subpathsOfDirectoryAtPath:error:` (page 412)
- `directoryContentsAtPath:` (page 398)
- `enumeratorAtPath:` (page 400)

Declared In

`NSFileManager.h`

subpathsOfDirectoryAtPath:error:

Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.

```
- (NSArray *)subpathsOfDirectoryAtPath:(NSString *)path error:(NSError **)error
```

Parameters

path

The path of the directory to list.

error

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

An array that contains `NSString` objects representing the filenames of the items in the directory specified by *path* and all its subdirectories recursively. If *path* is a symbolic link, `subpathsOfDirectoryAtPath:error:` traverses the link. Returns `nil` if it cannot get the device of the linked-to file.

Discussion

This list of directory contents goes very deep and hence is very useful for large file-system subtrees. The method skips “.” and “..”.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `subpathsAtPath:` (page 411)
- `directoryContentsAtPath:` (page 398)
- `enumeratorAtPath:` (page 400)

Declared In

`NSFileManager.h`

Delegate Methods

fileManager:shouldCopyItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldCopyItemAtPath:(NSString
    *)srcPath toPath:(NSString *)dstPath
```

Parameters

fileManager

The `NSFileManager` object that sent this message.

srcPath

The path or a file or directory that *manager* is about to attempt to copy.

dstPath

The path or a file or directory to which *manager* is about to attempt to copy.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

You can implement this method in your delegate to monitor file operations.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `copyItemAtPath:toPath:error:` (page 393)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 416)

Declared In

`NSFileManager.h`

fileManager:shouldLinkItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldLinkItemAtPath:(NSString
    *)srcPath toPath:(NSString *)dstPath
```

Parameters

fileManager

The `NSFileManager` object that sent this message.

srcPath

The path or a file or directory that *manager* is about to attempt to link.

dstPath

The path or a file or directory to which *manager* is about to attempt to link.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `linkItemAtPath:toPath:error:` (page 407)
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 416)

Declared In
`NSFileManager.h`

fileManager:shouldMoveItemAtPath:toPath:

An `NSFileManager` object sends this message immediately before attempting to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldMoveItemAtPath:(NSString
    *)srcPath toPath:(NSString *)dstPath
```

Parameters
fileManager
The `NSFileManager` object that sent this message.

srcPath
The path or a file or directory that *manager* is about to attempt to move.

dstPath
The path or a file or directory to which *manager* is about to attempt to move.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
You can implement this method in your delegate to monitor file operations.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `moveItemAtPath:toPath:error:` (page 408)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 417)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:

An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories.

```
- (BOOL)fileManager:(NSFileManager *)manager shouldProceedAfterError:(NSDictionary
    *)errorInfo
```

Parameters

manager

The file manager that sent this message.

errorInfo

A dictionary that contains two or three pieces of information (all `NSString` objects) related to the error:

Key	Value
`@"Path"`	The path related to the error (usually the source path)
`@"Error"`	A description of the error
`@"ToPath"`	The destination path (not all errors)

Return Value

`YES` if the operation (which is often continuous within a loop) should proceed, otherwise `NO`.

Discussion

An `NSFileManager` object, *manager*, sends this message for each error it encounters when copying, moving, removing, or linking files or directories. The return value is passed back to the invoker of `copyPath:toPath:handler:`, `movePath:toPath:handler:`, `removeFileAtPath:handler:`, or `linkPath:toPath:handler:`. If an error occurs and your handler has not implemented this method, the invoking method automatically returns `NO`.

The following implementation of `fileManager:shouldProceedAfterError:` displays the error string in an alert dialog and leaves it to the user whether to proceed or stop:

```
-(BOOL)fileManager:(NSFileManager *)manager
        shouldProceedAfterError:(NSDictionary *)errorInfo
{
    int result;
    result = NSRunAlertPanel(@"Gumby App", @"File  operation error:
            %@ with file: %@", @"Proceed", @"Stop",  NULL,
            [errorInfo objectForKey:@"Error"],
            [errorInfo objectForKey:@"Path"]);

    if (result == NSAlertDefaultReturn)
        return YES;
    else
        return NO;
}
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`- fileManager:willProcessPath:` (page 419)

Declared In

`NSFileManager.h`

fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:

An NSFileManager object sends this message if an error occurs during an attempt to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error copyingItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath
```

Parameters

fileManager

The NSFileManager object that sent this message.

error

The error that occurred during the attempt to copy.

srcPath

The path or a file or directory that *manager* is attempting to copy.

dstPath

The path or a file or directory to which *manager* is attempting to copy.

Return Value

YES if the operation should proceed, otherwise NO.

Discussion

You can implement this method in your delegate to monitor file operations.

Availability

Available in iPhone OS 2.0 and later.

See Also

- copyItemAtPath:toPath:error: (page 393)
- fileManager:shouldCopyItemAtPath:toPath: (page 413)

Declared In

NSFileManager.h

fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:

An NSFileManager object sends this message if an error occurs during an attempt to link to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    linkingItemAtPath:(NSString *)srcPath
    toPath:(NSString *)dstPath
```

Parameters

fileManager

The NSFileManager object that sent this message.

error

The error that occurred during the attempt to link.

srcPath

The path or a file or directory that *manager* is attempting to link.

dstPath
The path or a file or directory to which *manager* is attempting to link.

Return Value
YES if the operation should proceed, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- linkItemAtPath:toPath:error: (page 407)
- fileManager:shouldLinkItemAtPath:toPath: (page 413)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:movingItemAtPath:toPath:

An NSFileManager object sends this message if an error occurs during an attempt to move to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error movingItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath;
```

Parameters
fileManager
The NSFileManager object that sent this message.

error
The error that occurred during the attempt to move.

srcPath
The path or a file or directory that *manager* is attempting to move.

dstPath
The path or a file or directory to which *manager* is attempting to move.

Return Value
YES if the operation should proceed, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- moveItemAtPath:toPath:error: (page 408)
- fileManager:shouldMoveItemAtPath:toPath: (page 414)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:removingItemAtPath:

An NSFileManager object sends this message if an error occurs during an attempt to delete a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldProceedAfterError:(NSError *)error
    removingItemAtPath:(NSString *)path
```

Parameters

fileManager

The NSFileManager object that sent this message.

error

The error that occurred during the attempt to copy.

path

The path or a file or directory that *manager* is attempting to delete.

Return Value

YES if the operation should proceed, otherwise NO.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeItemAtPath:error: (page 409)
- fileManager:shouldRemoveItemAtPath: (page 418)

Declared In

NSFileManager.h

fileManager:shouldRemoveItemAtPath:

An NSFileManager object sends this message immediately before attempting to copy to a given path.

```
- (BOOL)fileManager:(NSFileManager *)fileManager
    shouldRemoveItemAtPath:(NSString *)path
```

Parameters

fileManager

The NSFileManager object that sent this message.

path

The path or a file or directory that *manager* is about to attempt to delete.

Return Value

YES if the operation should proceed, otherwise NO.

Discussion

You can implement this method in your delegate to monitor file operations.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeItemAtPath:error: (page 409)
- fileManager:shouldProceedAfterError:removingItemAtPath: (page 417)

Declared In

NSFileManager.h

fileManager:willProcessPath:

An `NSFileManager` object sends this message to a handler immediately before attempting to move, copy, rename, or delete, or before attempting to link to a given path.

```
- (void)fileManager:(NSFileManager *)manager willProcessPath:(NSString *)path
```

Parameters

manager

The `NSFileManager` object that sent this message.

path

The path or a file or directory that *manager* is about to attempt to move, copy, rename, delete, or link to.

Discussion

You can implement this method in your handler to monitor file operations.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSFileManager.h`

Constants

File Attribute Keys

These keys access file attribute values contained in `NSDictionary` objects used by `changeFileAttributes:atPath:` (page 390), `fileAttributesAtPath:traverseLink:` (page 401), `createDirectoryAtPath:attributes:` (page 394), and `createFileAtPath:contents:attributes:` (page 395).

```
NSString *NSFileType;
NSString *NSFileTypeDirectory;
NSString *NSFileTypeRegular;
NSString *NSFileTypeSymbolicLink;
NSString *NSFileTypeSocket;
NSString *NSFileTypeCharacterSpecial;
NSString *NSFileTypeBlockSpecial;
NSString *NSFileTypeUnknown;
NSString *NSFileSize;
NSString *NSFileModificationDate;
NSString *NSFileReferenceCount;
NSString *NSFileDeviceIdentifier;
NSString *NSFileOwnerAccountName;
NSString *NSFileGroupOwnerAccountName;
NSString *NSFilePosixPermissions;
NSString *NSFileSystemNumber;
NSString *NSFileSystemFileNumber;
NSString *NSFileExtensionHidden;
NSString *NSFileHFSCreatorCode;
NSString *NSFileHFSTypeCode;
NSString *NSFileImmutable;
NSString *NSFileAppendOnly;
NSString *NSFileCreationDate;
NSString *NSFileOwnerAccountID;
NSString *NSFileGroupOwnerAccountID;
NSString *NSFileBusy;
```

Constants

`NSFileAppendOnly`

The key in a file attribute dictionary whose value indicates whether the file is read-only.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileBusy`

The key in a file attribute dictionary whose value indicates whether the file is busy.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileCreationDate`

The key in a file attribute dictionary whose value indicates the file's creation date.

The corresponding value is an `NSDate` object.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileOwnerAccountName`

The key in a file attribute dictionary whose value indicates the name of the file's owner.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileGroupOwnerAccountName`
The key in a file attribute dictionary whose value indicates the group name of the file's owner.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileDeviceIdentifier`
The key in a file attribute dictionary whose value indicates the identifier for the device on which the file resides.

The corresponding value is an `NSNumber` object containing an `unsigned long`.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileExtensionHidden`
The key in a file attribute dictionary whose value indicates whether the file's extension is hidden.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileGroupOwnerAccountID`
The key in a file attribute dictionary whose value indicates the file's group ID.

The corresponding value is an `NSNumber` object containing an `unsigned long`.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileHFSCreatorCode`
The key in a file attribute dictionary whose value indicates the file's HFS creator code.

The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileHFSTypeCode`
The key in a file attribute dictionary whose value indicates the file's HFS type code.

The corresponding value is an `NSNumber` object containing an `unsigned long`. See HFS File Types for possible values.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileImmutable`
The key in a file attribute dictionary whose value indicates whether the file is mutable.

The corresponding value is an `NSNumber` object containing a Boolean value.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

`NSFileModificationDate`
The key in a file attribute dictionary whose value indicates the file's last modified date.
The corresponding value is an `NSDate` object.
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFileOwnerAccountID`
The key in a file attribute dictionary whose value indicates the file's owner's account ID.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFilePosixPermissions`
The key in a file attribute dictionary whose value indicates the file's Posix permissions.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFileReferenceCount`
The key in a file attribute dictionary whose value indicates the file's reference count.
The corresponding value is an `NSNumber` object containing an `unsigned long`.
The number specifies the number of hard links to a file.
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFileSize`
The key in a file attribute dictionary whose value indicates the file's size in bytes.
The corresponding value is an `NSNumber` object containing an `unsigned long long`.

Important: If the file has a resource fork, the returned value does *not* include the size of the resource fork.

Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFileSystemFileNumber`
The key in a file attribute dictionary whose value indicates the file's filesystem file number.
The corresponding value is an `NSNumber` object containing an `unsigned long`. The value corresponds to the value of `st_ino`, as returned by `stat`(2).
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

`NSFileType`
The key in a file attribute dictionary whose value indicates the file's type.
The corresponding value is an `NSString` object (see below for possible values).
Available in iPhone OS 2.0 and later.
Declared in `NSFileManager.h`

Discussion
`NSFileDeviceIdentifier` is used to access the identifier of a remote device.

Declared In
NSFileManager.h

File Type Attribute Keys

These strings the possible values for the NSFileType attribute key contained in the NSDictionary object returned from NSFileManager's fileAttributesAtPath:traverseLink: (page 401).

```
extern NSString *NSFileTypeDirectory;
extern NSString *NSFileTypeRegular;
extern NSString *NSFileTypeSymbolicLink;
extern NSString *NSFileTypeSocket;
extern NSString *NSFileTypeCharacterSpecial;
extern NSString *NSFileTypeBlockSpecial;
extern NSString *NSFileTypeUnknown;
```

Constants

NSFileTypeDirectory
Directory
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeRegular
Regular file
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeSymbolicLink
Symbolic link
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeSocket
Socket
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeCharacterSpecial
Character special file
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeBlockSpecial
Block special file
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

NSFileTypeUnknown
Unknown
Available in iPhone OS 2.0 and later.
Declared in NSFileManager.h

Declared In
NSFileManager.h

File-System Attribute Keys

Keys to access the file attribute values contained in the NSDictionary object returned from NSFileManager's fileSystemAttributesAtPath: (page 403) method.

```
extern NSString *NSFileSystemSize;
extern NSString *NSFileSystemFreeSize;
extern NSString *NSFileSystemNodes;
extern NSString *NSFileSystemFreeNodes;
extern NSString *NSFileSystemNumber;
```

Constants

NSFileSystemSize
: The key in a file system attribute dictionary whose value indicates the size of the file system.

  The corresponding value is an NSNumber object that specifies the size of the file system in bytes. The value is determined by statfs().

  Available in iPhone OS 2.0 and later.

  Declared in NSFileManager.h

NSFileSystemFreeSize
: The key in a file system attribute dictionary whose value indicates the amount of free space on the file system.

  The corresponding value is an NSNumber object that specifies the amount of free space on the file system in bytes. The value is determined by statfs().

  Available in iPhone OS 2.0 and later.

  Declared in NSFileManager.h

NSFileSystemNodes
: The key in a file system attribute dictionary whose value indicates the number of nodes in the file system.

  The corresponding value is an NSNumber object that specifies the number of nodes in the file system.

  Available in iPhone OS 2.0 and later.

  Declared in NSFileManager.h

NSFileSystemFreeNodes
: The key in a file system attribute dictionary dictionary whose value indicates the number of free nodes in the file system.

  The corresponding value is an NSNumber object that specifies the number of free nodes in the file system.

  Available in iPhone OS 2.0 and later.

  Declared in NSFileManager.h

`NSFileSystemNumber`

The key in a file system attribute dictionary dictionary whose value indicates the filesystem number of the file system.

The corresponding value is an `NSNumber` object that specifies the filesystem number of the file system. The value corresponds to the value of `st_dev`, as returned by `stat`(2).

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

Declared In
`NSFileManager.h`

Resource Fork Support

Specifies the version of the Foundation framework in which `NSFileManager` first supported resource forks.

```
#define NSFoundationVersionWithFileManagerResourceForkSupport 412
```

Constants
`NSFoundationVersionWithFileManagerResourceForkSupport`

The version of the Foundation framework in which `NSFileManager` first supported resource forks.

Available in iPhone OS 2.0 and later.

Declared in `NSFileManager.h`

Declared In
`NSFileManager.h`

NSFormatter Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSFormatter.h
Companion guide:	Data Formatting Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSFormatter` is an abstract class that declares an interface for objects that create, interpret, and validate the textual representation of cell contents. The Foundation framework provides two concrete subclasses of `NSFormatter` to generate these objects: `NSNumberFormatter` and `NSDateFormatter`.

Subclassing Notes

`NSFormatter` is intended for subclassing. A custom formatter can restrict the input and enhance the display of data in novel ways. For example, you could have a custom formatter that ensures that serial numbers entered by a user conform to predefined formats. Before you decide to create a custom formatter, make sure that you cannot configure the public subclasses `NSDateFormatter` and `NSNumberFormatter` to satisfy your requirements.

For instructions on how to create your own custom formatter, see Creating a Custom Formatter.

Tasks

Textual Representation of Cell Content

`- stringForObjectValue:` (page 433)
The default implementation of this method raises an exception.

`- attributedStringForObjectValue:withDefaultAttributes:` (page 429)
The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.

`- editingStringForObjectValue:` (page 429)
The default implementation of this method invokes `stringForObjectValue:` (page 433).

Object Equivalent to Textual Representation

`- getObjectValue:forString:errorDescription:` (page 430)
The default implementation of this method raises an exception.

Dynamic Cell Editing

`- isPartialStringValid:newEditingString:errorDescription:` (page 431)
Returns a Boolean value that indicates whether a partial string is valid.

`- isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 432)
This method should be implemented in subclasses that want to validate user changes to a string in a field, where the user changes are not necessarily at the end of the string, and preserve the selection (or set a different one, such as selecting the erroneous part of the string the user has typed).

Instance Methods

attributedStringForObjectValue:withDefaultAttributes:

The default implementation returns `nil` to indicate that the formatter object does not provide an attributed string.

```
- (NSAttributedString *)attributedStringForObjectValue:(id)anObject
    withDefaultAttributes:(NSDictionary *)attributes
```

Parameters

anObject

The object for which a textual representation is returned.

attributes

The default attributes to use for the returned attributed string.

Return Value

An attributed string that represents *anObject*.

Discussion

When implementing a subclass, return an `NSAttributedString` object if the string for display should have some attributes. For instance, you might want negative values in a financial application to appear in red text. Invoke your implementation of `stringForObjectValue:` (page 433) to get the non-attributed string, then create an `NSAttributedString` object with it (see `initWithString:`). Use the *attributes* default dictionary to reset the attributes of the string when a change in value warrants it (for example, a negative value becomes positive) For information on creating attributed strings, see *Attributed Strings Programming Guide*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- editingStringForObjectValue:` (page 429)

Declared In

`NSFormatter.h`

editingStringForObjectValue:

The default implementation of this method invokes `stringForObjectValue:` (page 433).

```
- (NSString *)editingStringForObjectValue:(id)anObject
```

Parameters

anObject

The object for which to return an editing string.

Return Value

An `NSString` object that is used for editing the textual representation of *anObject*.

Discussion
When implementing a subclass, override this method only when the string that users see and the string that they edit are different. In your implementation, return an `NSString` object that is used for editing, following the logic recommended for implementing `stringForObjectValue:` (page 433). As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- attributedStringForObjectValue:withDefaultAttributes:` (page 429)

Declared In
`NSFormatter.h`

getObjectValue:forString:errorDescription:
The default implementation of this method raises an exception.

```
- (BOOL)getObjectValue:(id *)anObject forString:(NSString *)string
    errorDescription:(NSString **)error
```

Parameters
anObject
: If conversion is successful, upon return contains the object created from *string*.

string
: The string to parse.

error
: If non-`nil`, if there is a error during the conversion, upon return contains an `NSString` object that describes the problem.

Return Value
`YES` if the conversion from string to cell content object was successful, otherwise `NO`.

Discussion
When implementing a subclass, return by reference the object *anObject* after creating it from *string*. Return `YES` if the conversion is successful. If you return `NO`, also return by indirection (in *error*) a localized user-presentable `NSString` object that explains the reason why the conversion failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToFormatString:errorDescription:`. However, if *error* is `nil`, the sender is not interested in the error description, and you should not attempt to assign one.

The following example (which is paired with the example given in `stringForObjectValue:` (page 433)) converts a string representation of a dollar amount that includes the dollar sign; it uses an `NSScanner` instance to convert this amount to a float after stripping out the initial dollar sign.

```
- (BOOL)getObjectValue:(id *)obj forString:(NSString *)string
errorDescription:(NSString  **)error
{
    float floatResult;
    NSScanner *scanner;
    BOOL returnValue = NO;
```

```
    scanner = [NSScanner scannerWithString: string];
    [scanner scanString: @"$" intoString: NULL];    //ignore  return value
    if ([scanner scanFloat:&floatResult] && ([scanner isAtEnd])) {
        returnValue = YES;
        if (obj)
            *obj = [NSNumber numberWithFloat:floatResult];
    } else {
        if (error)
            *error = NSLocalizedString(@"Couldn't convert  to float", @"Error
converting");
    }
    return returnValue;
}
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringForObjectValue:` (page 433)

Declared In
`NSFormatter.h`

isPartialStringValid:newEditingString:errorDescription:

Returns a Boolean value that indicates whether a partial string is valid.

```
- (BOOL)isPartialStringValid:(NSString *)partialString newEditingString:(NSString
    **)newString errorDescription:(NSString **)error
```

Parameters
partialString
The text currently in a cell.

newString
If *partialString* needs to be modified, upon return contains the replacement string.

error
If non-`nil`, if validation fails contains an `NSString` object that describes the problem.

Return Value
`YES` if *partialString* is an acceptable value, otherwise `NO`.

Discussion
This method is invoked each time the user presses a key while the cell has the keyboard focus—it lets you verify and edit the cell text as the user types it.

In a subclass implementation, evaluate *partialString* according to the context, edit the text if necessary, and return by reference any edited string in *newString*. Return `YES` if *partialString* is acceptable and `NO` if *partialString* is unacceptable. If you return `NO` and *newString* is `nil`, the cell displays *partialString* minus the last character typed. If you return `NO`, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:`. The selection range will always be set to the end of the text if replacement occurs.

This method is a compatibility method. If a subclass overrides this method and does not override `isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 432), this method will be called as before (`isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 432) just calls this one by default).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFormatter.h`

isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:

This method should be implemented in subclasses that want to validate user changes to a string in a field, where the user changes are not necessarily at the end of the string, and preserve the selection (or set a different one, such as selecting the erroneous part of the string the user has typed).

```
- (BOOL)isPartialStringValid:(NSString **)partialStringPtr
    proposedSelectedRange:(NSRangePointer)proposedSelRangePtr
    originalString:(NSString *)origString originalSelectedRange:(NSRange)origSelRange
     errorDescription:(NSString **)error
```

Parameters
partialStringPtr
The new string to validate.

proposedSelRangePtr
The selection range that will be used if the string is accepted or replaced.

origString
The original string, before the proposed change.

origSelRange
The selection range over which the change is to take place.

error
If non-`nil`, if validation fails contains an `NSString` object that describes the problem.

Return Value
`YES` if *partialStringPtr* is acceptable, otherwise `NO`.

Discussion
In a subclass implementation, evaluate *partialString* according to the context. Return `YES` if *partialStringPtr* is acceptable and `NO` if *partialStringPtr* is unacceptable. Assign a new string to *partialStringPtr* and a new range to *proposedSelRangePtr* and return `NO` if you want to replace the string and change the selection range. If you return `NO`, you can also return by indirection an `NSString` object (in *error*) that explains the reason why the validation failed; the delegate (if any) of the `NSControl` object managing the cell can then respond to the failure in `control:didFailToValidatePartialString:errorDescription:`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `isPartialStringValid:newEditingString:errorDescription:` (page 431)

Declared In
`NSFormatter.h`

stringForObjectValue:

The default implementation of this method raises an exception.

```
- (NSString *)stringForObjectValue:(id)anObject
```

Parameters
anObject
The object for which a textual representation is returned.

Return Value
An `NSString` object that textually represents *object* for display. Returns `nil` if *object* is not of the correct class.

Discussion
When implementing a subclass, return the `NSString` object that textually represents the cell's object for display and—if `editingStringForObjectValue:` (page 429) is unimplemented—for editing. First test the passed-in object to see if it's of the correct class. If it isn't, return `nil`; but if it is of the right class, return a properly formatted and, if necessary, localized string. (See the specification of the `NSString` class for formatting and localizing details.)

The following implementation (which is paired with the `getObjectValue:forString:errorDescription:` (page 430) example above) prefixes a two-digit float representation with a dollar sign:

```
- (NSString *)stringForObjectValue:(id)anObject
{
    if (![anObject isKindOfClass:[NSNumber class]]) {
        return nil;
    }
    return [NSString stringWithFormat:@"$%.2f", [anObject  floatValue]];
}
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `attributedStringForObjectValue:withDefaultAttributes:` (page 429)
- `editingStringForObjectValue:` (page 429)
- `getObjectValue:forString:errorDescription:` (page 430)

Declared In
`NSFormatter.h`

NSHTTPCookie Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSHTTPCookie.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An NSHTTPCookie object represents an HTTP cookie. It's an immutable object initialized from a dictionary containing the cookie attributes.

Two versions of cookies are supported:

- Version 0: This version refers to "traditional" or "old-style" cookies, the original cookie format defined by Netscape. The majority of cookies encountered are in this format.
- Version 1: This version refers to cookies as defined in RFC 2965, HTTP State Management Mechanism.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 1250)

Tasks

Create Cookie Instances

`+ cookiesWithResponseHeaderFields:forURL:` (page 437)
Returns an array of NSHTTPCookie objects corresponding to the provided response header fields for the provided URL.

`+ cookieWithProperties:` (page 437)
Creates and initializes an NSHTTPCookie object using the provided properties.

`- initWithProperties:` (page 440)
Returns an initialized `NSHTTPCookie` object using the provided properties.

Convert Cookies to Request Headers

`+ requestHeaderFieldsWithCookies:` (page 438)
Returns a dictionary of header fields corresponding to a provided array of cookies.

Getting Cookie Properties

`- comment` (page 438)
Returns the receiver's comment string.

`- commentURL` (page 439)
Returns the receiver's comment URL.

`- domain` (page 439)
Returns the domain of the receiver's cookie.

`- expiresDate` (page 439)
Returns the receiver's expiration date.

`- isSecure` (page 440)
Returns whether his cookie should only be sent over secure channels.

`- isSessionOnly` (page 440)
Returns whether the receiver should be discarded at the end of the session (regardless of expiration date).

`- name` (page 441)
Returns the receiver's name.

`- path` (page 441)
Returns the receiver's path.

- portList (page 441)
  Returns the receiver's port list.
- properties (page 442)
  Returns the receiver's cookie properties.
- value (page 442)
  Returns the receiver's value.
- version (page 442)
  Returns the receiver's version.

Class Methods

cookiesWithResponseHeaderFields:forURL:

Returns an array of NSHTTPCookie objects corresponding to the provided response header fields for the provided URL.

```
+ (NSArray *)cookiesWithResponseHeaderFields:(NSDictionary *)headerFields
    forURL:(NSURL *)theURL
```

Parameters

headerFields
  The header fields used to create the NSHTTPCookie objects.

theURL
  The URL associated with the created cookies.

Return Value

The array of newly created cookies.

Discussion

This method will ignore irrelevant header fields in *headerFields*, allowing dictionaries to contain additional data.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSHTTPCookie.h

cookieWithProperties:

Creates and initializes an NSHTTPCookie object using the provided properties.

```
+ (id)cookieWithProperties:(NSDictionary *)properties
```

Parameters

properties
  The properties for the new cookie object, expressed as key value pairs.

Return Value

The newly created cookie object. Returns nil if the provided properties are invalid.

Discussion
See "Constants" (page 443) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithProperties: (page 440)

Declared In
NSHTTPCookie.h

requestHeaderFieldsWithCookies:

Returns a dictionary of header fields corresponding to a provided array of cookies.

```
+ (NSDictionary *)requestHeaderFieldsWithCookies:(NSArray *)cookies
```

Parameters

cookies
The cookies from which the header fields are created.

Return Value
The dictionary of header fields created from the provided cookies. This dictionary can be used to add cookies to a request.

Discussion
See "Constants" (page 443) for details on the header field keys and values in the returned dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

Instance Methods

comment

Returns the receiver's comment string.

```
- (NSString *)comment
```

Return Value
The receiver's comment string or `nil` if the cookie has no comment. This string is suitable for presentation to the user, explaining the contents and purpose of this cookie.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

commentURL

Returns the receiver's comment URL.

```
- (NSURL *)commentURL
```

Return Value
The receiver's comment URL or `nil` if the cookie has none. This value specifies a URL which is suitable for presentation to the user as a link for further information about this cookie.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

domain

Returns the domain of the receiver's cookie.

```
- (NSString *)domain
```

Return Value
The domain of the receiver's cookie.

Discussion
If the domain does not start with a dot, then the cookie will only be sent to the exact host specified by the domain. If the domain does start with a dot, then the cookie will be sent to other hosts in that domain as well, subject to certain restrictions. See RFC 2965 for more detail.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

expiresDate

Returns the receiver's expiration date.

```
- (NSDate *)expiresDate
```

Return Value
The receiver's expiration date, or `nil` if there is no specific expiration date such as in the case of "session-only" cookies. The expiration date is the date when the cookie should be deleted.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

initWithProperties:

Returns an initialized NSHTTPCookie object using the provided properties.

```
- (id)initWithProperties:(NSDictionary *)properties
```

Parameters

properties

The properties for the new cookie object, expressed as key value pairs.

Return Value
The initialized cookie object. Returns nil if the provided properties are invalid.

Discussion
See "Constants" (page 443) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ cookieWithProperties: (page 437)

Declared In
NSHTTPCookie.h

isSecure

Returns whether his cookie should only be sent over secure channels.

```
- (BOOL)isSecure
```

Return Value
YES if this cookie should only be sent over secure channels, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookie.h

isSessionOnly

Returns whether the receiver should be discarded at the end of the session (regardless of expiration date).

```
- (BOOL)isSessionOnly
```

Return Value
`YES` if the receiver should be discarded at the end of the session (regardless of expiration date), otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

name
Returns the receiver’s name.

```
- (NSString *)name
```

Return Value
The receiver's name.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

path
Returns the receiver’s path.

```
- (NSString *)path
```

Return Value
The receiver's path.

Discussion
The cookie will be sent with requests for this path in the cookie's domain, and all paths that have this prefix. A path of “/” means the cookie will be sent for all URLs in the domain.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

portList
Returns the receiver's port list.

```
- (NSArray *)portList
```

Return Value
The list of ports for the cookie, returned as an array of `NSNumber` objects containing integers. If the cookie has no port list this method returns `nil` and the cookie will be sent to any port. Otherwise, the cookie is only sent to ports specified in the port list.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

properties

Returns the receiver's cookie properties.

```
- (NSDictionary *)properties
```

Return Value
A dictionary representation of the receiver's cookie properties.

Discussion
This dictionary can be used with `initWithProperties:` (page 440) or `cookieWithProperties:` (page 437) to create an equivalent `NSHTTPCookie` object.

See `initWithProperties:` (page 440) for more information on the constraints imposed on the *`properties`* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

value

Returns the receiver's value.

```
- (NSString *)value
```

Return Value
The receiver's value.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

version

Returns the receiver's version.

```
- (NSUInteger)version
```

Return Value
The receiver's version. Version 0 maps to “old-style” Netscape cookies. Version 1 maps to RFC 2965 cookies.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookie.h`

Constants

HTTP Cookie Property Keys
These constants define the supported keys in a dictionary containing cookie attributes.

```
extern NSString *NSHTTPCookieComment;
extern NSString *NSHTTPCookieCommentURL;
extern NSString *NSHTTPCookieDiscard;
extern NSString *NSHTTPCookieDomain;
extern NSString *NSHTTPCookieExpires;
extern NSString *NSHTTPCookieMaximumAge;
extern NSString *NSHTTPCookieName;
extern NSString *NSHTTPCookieOriginURL;
extern NSString *NSHTTPCookiePath;
extern NSString *NSHTTPCookiePort;
extern NSString *NSHTTPCookieSecure;
extern NSString *NSHTTPCookieValue;
extern NSString *NSHTTPCookieVersion;
```

Constants
`NSHTTPCookieComment`

An `NSString` object containing the comment for the cookie.

Only valid for Version 1 cookies and later. This header field is optional.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

`NSHTTPCookieCommentURL`

An `NSURL` object or `NSString` object containing the comment URL for the cookie.

Only valid for Version 1 cookies or later. This header field is optional.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

`NSHTTPCookieDiscard`

An `NSString` object stating whether the cookie should be discarded at the end of the session.

String value must be either “TRUE” or “FALSE”. This header field is optional. Default is “FALSE”, unless this is cookie is version 1 or greater and a value for `NSHTTPCookieMaximumAge` is not specified, in which case it is assumed “TRUE”.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

`NSHTTPCookieDomain`
: An `NSString` object containing the domain for the cookie.

  A value must be specified for either `NSHTTPCookieDomain` or `NSHTTPCookieOriginURL`. If this header field is missing the domain is inferred from the value for `NSHTTPCookieOriginURL`.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookieExpires`
: An `NSDate` object or `NSString` object specifying the expiration date for the cookie.

  This header field is only used for Version 0 cookies. This header field is optional.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookieMaximumAge`
: An `NSString` object containing an integer value stating how long in seconds the cookie should be kept, at most.

  Only valid for Version 1 cookies and later. Default is “0”. This field is optional.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookieName`
: An `NSString` object containing the name of the cookie. This field is required.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookieOriginURL`
: An NSURL or `NSString` object containing the URL that set this cookie.

  A value must be specified for either `NSHTTPCookieDomain` or `NSHTTPCookieOriginURL`.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookiePath`
: An `NSString` object containing the path for the cookie.

  Inferred from the value for `NSHTTPCookieOriginURL` if not provided. Default is “/”. This header field is optional.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookiePort`
: An `NSString` object containing comma-separated integer values specifying the ports for the cookie.

  Only valid for Version 1 cookies or later. The default value is an empty string (““). This header field is optional.

  Available in iPhone OS 2.0 and later.

  Declared in `NSHTTPCookie.h`

`NSHTTPCookieSecure`

An `NSString` object stating whether the cookie should be transmitted only over secure channels.

String value must be either “TRUE” or “FALSE”. Default is “FALSE”. This header field is optional.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

`NSHTTPCookieValue`

An `NSString` object containing the value of the cookie.

This header field is required.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

`NSHTTPCookieVersion`

An `NSString` object that specifies the version of the cookie.

Must be either “0” or “1”. The default is “0”. This header field is optional.

Available in iPhone OS 2.0 and later.

Declared in `NSHTTPCookie.h`

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In

`NSHTTPCookie.h`

NSHTTPCookieStorage Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSHTTPCookieStorage.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSHTTPCookieStorage` implements a singleton object (shared instance) that manages the shared cookie storage. These cookies are shared among all applications and are kept in sync cross-process.

Note: Changes made to the cookie accept policy will affect all currently running applications using the cookie storage.

Tasks

Getting the Shared Cookie Storage Object

`+ sharedHTTPCookieStorage` (page 448)
Returns the shared cookie storage instance.

Getting and Setting the Cookie Accept Policy

`- cookieAcceptPolicy` (page 449)
Returns the receiver's cookie accept policy.

`- setCookieAcceptPolicy:` (page 451)
Sets the cookie accept policy of the receiver

Adding and Removing Cookies

`- cookies` (page 449)
Returns the receiver's cookies.

`- cookiesForURL:` (page 449)
Returns all the receiver's cookies that will be sent to a specified URL.

`- deleteCookie:` (page 450)
Deletes the specified cookie from the receiver.

`- setCookie:` (page 450)
Stores a specified cookie in the receiver if the receiver's cookie accept policy permits.

`- setCookies:forURL:mainDocumentURL:` (page 451)
Adds an array of cookies to the receiver if the receiver's cookie acceptance policy permits.

Class Methods

sharedHTTPCookieStorage

Returns the shared cookie storage instance.

```
+ (NSHTTPCookieStorage *)sharedHTTPCookieStorage
```

Return Value

The shared cookie storage instance.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookieStorage.h

Instance Methods

cookieAcceptPolicy

Returns the receiver's cookie accept policy.

```
- (NSHTTPCookieAcceptPolicy)cookieAcceptPolicy
```

Return Value
The receiver's cookie accept policy. The default cookie accept policy is NSHTTPCookieAcceptPolicyAlways.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setCookieAcceptPolicy: (page 451)

Declared In
NSHTTPCookieStorage.h

cookies

Returns the receiver's cookies.

```
- (NSArray *)cookies
```

Return Value
An array containing all of the receiver's cookies.

Availability
Available in iPhone OS 2.0 and later.

See Also
- cookiesForURL: (page 449)

Declared In
NSHTTPCookieStorage.h

cookiesForURL:

Returns all the receiver's cookies that will be sent to a specified URL.

```
- (NSArray *)cookiesForURL:(NSURL *)theURL
```

Parameters

theURL

The URL to filter on.

Return Value

An array of cookies whose URL matches the provided URL.

Discussion

An application can use `NSHTTPCookie`'s `requestHeaderFieldsWithCookies:` (page 438) method to turn this array into a set of header fields to add to an `NSMutableURLRequest` object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- cookies` (page 449)

Declared In

`NSHTTPCookieStorage.h`

deleteCookie:

Deletes the specified cookie from the receiver.

```
- (void)deleteCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to delete.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSHTTPCookieStorage.h`

setCookie:

Stores a specified cookie in the receiver if the receiver's cookie accept policy permits.

```
- (void)setCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to store.

Discussion

The cookie will replace an existing cookie with the same name, domain and path, if one exists in the cookie storage. This method will accept the cookie only if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyAlways` or `NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`. The cookie will be ignored if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyNever`.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookieStorage.h

setCookieAcceptPolicy:

Sets the cookie accept policy of the receiver

```
- (void)setCookieAcceptPolicy:(NSHTTPCookieAcceptPolicy)aPolicy
```

Parameters

aPolicy
The new cookie accept policy.

Discussion
The default cookie accept policy is NSHTTPCookieAcceptPolicyAlways. Changing the cookie policy will effect all currently running applications using the cookie storage.

Availability
Available in iPhone OS 2.0 and later.

See Also
- cookieAcceptPolicy (page 449)

Declared In
NSHTTPCookieStorage.h

setCookies:forURL:mainDocumentURL:

Adds an array of cookies to the receiver if the receiver's cookie acceptance policy permits.

```
- (void)setCookies:(NSArray *)cookies forURL:(NSURL *)theURL mainDocumentURL:(NSURL
    *)mainDocumentURL
```

Parameters

cookies
The cookies to add.

theURL
The URL associated with the added cookies.

mainDocumentURL
The URL of the main HTML document for the top-level frame, if known. Can be nil. This URL is used to determine if the cookie should be accepted if the cookie accept policy is NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain.

Discussion
The cookies will replace existing cookies with the same name, domain, and path, if one exists in the cookie storage. The cookie will be ignored if the receiver's cookie accept policy is NSHTTPCookieAcceptPolicyNever.

To store cookies from a set of response headers, an application can use cookiesWithResponseHeaderFields:forURL: (page 437) passing a header field dictionary and then use this method to store the resulting cookies in accordance with the receiver's cookie acceptance policy.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookieStorage.h

Constants

NSHTTPCookieAcceptPolicy

NSHTTPCookieAcceptPolicy specifies the cookie acceptance policies implemented by the NSHTTPCookieStorage class.

```
typedef enum {
    NSHTTPCookieAcceptPolicyAlways,
    NSHTTPCookieAcceptPolicyNever,
    NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain
} NSHTTPCookieAcceptPolicy;
```

Constants

NSHTTPCookieAcceptPolicyAlways
Accept all cookies. This is the default cookie accept policy.
Available in iPhone OS 2.0 and later.
Declared in NSHTTPCookieStorage.h

NSHTTPCookieAcceptPolicyNever
Reject all cookies.
Available in iPhone OS 2.0 and later.
Declared in NSHTTPCookieStorage.h

NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain
Accept cookies only from the main document domain.
Available in iPhone OS 2.0 and later.
Declared in NSHTTPCookieStorage.h

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSHTTPCookieStorage.h

Notifications

NSHTTPCookieManagerCookiesChangedNotification

This notification is posted when the cookies stored in the NSHTTPCookieStorage instance have changed. Since cookies are shared among applications, this notification can be sent in response to another application's actions.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookieStorage.h`

NSHTTPCookieManagerAcceptPolicyChangedNotification

This notification is posted when the acceptance policy of the `NSHTTPCookieStorage` instance has changed. Since cookies are shared among applications, this notification can be sent in response to another application's actions.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSHTTPCookieStorage.h`

NSHTTPURLResponse Class Reference

Inherits from:	NSURLResponse : NSObject
Conforms to:	NSCopying (NSURLResponse) NSCoding (NSURLResponse) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLResponse.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An NSHTTPURLResponse object represents a response to an HTTP URL load request. It's a subclass of NSURLResponse that provides methods for accessing information specific to HTTP protocol responses.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

Tasks

Getting HTTP Response Headers

`- allHeaderFields` (page 457)
Returns all the HTTP header fields of the receiver.

Getting Response Status Code

`+ localizedStringForStatusCode:` (page 456)
Returns a localized string corresponding to a specified HTTP status code.

`- statusCode` (page 457)
Returns the receiver's HTTP status code.

Class Methods

localizedStringForStatusCode:

Returns a localized string corresponding to a specified HTTP status code.

```
+ (NSString *)localizedStringForStatusCode:(NSInteger)statusCode
```

Parameters

statusCode
The HTTP status code.

Return Value

A localized string suitable for displaying to users that describes the specified status code.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- statusCode` (page 457)

Declared In
NSURLResponse.h

Instance Methods

allHeaderFields

Returns all the HTTP header fields of the receiver.

```
- (NSDictionary *)allHeaderFields
```

Return Value
A dictionary containing all the HTTP header fields of the receiver. By examining this dictionary clients can see the "raw" header information returned by the HTTP server.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLResponse.h

statusCode

Returns the receiver's HTTP status code.

```
- (NSInteger)statusCode
```

Return Value
The receiver's HTTP status code.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ localizedStringForStatusCode: (page 456)

Declared In
NSURLResponse.h

NSIndexPath Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSIndexPath.h

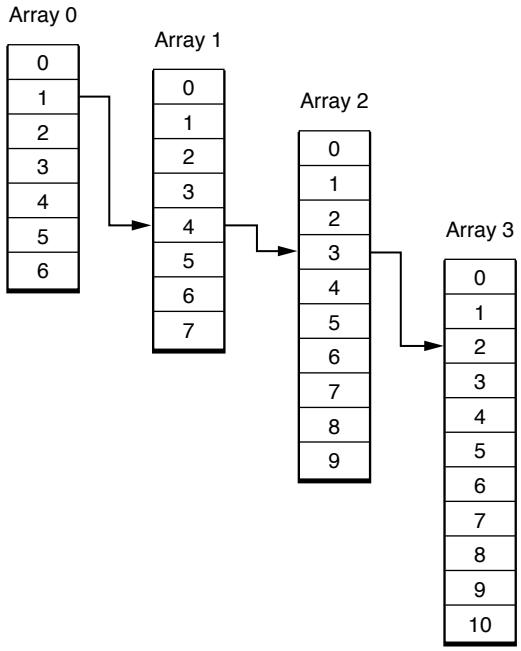
Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSIndexPath` class represents the path to a specific node in a tree of nested array collections. This path is known as an **index path**.

Each index in an index path represents the index into an array of children from one node in the tree to another, deeper, node. For example, the index path `1.4.3.2` specifies the path shown in Figure 30-1.

Figure 30-1 Index path `1.4.3.2`



`NSIndexPath` objects are uniqued and shared. If an index path containing the specified index or indexes already exists, that object is returned instead of a new instance.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

Tasks

Creating Index Paths

`+ indexPathWithIndex:` (page 461)
Creates an one-node index path.

`+ indexPathWithIndexes:length:` (page 462)
Creates an index path with one or more nodes.

`- initWithIndex:` (page 464)
Initializes an allocated `NSIndexPath` (page 459) object with a one-node index path.

- initWithIndexes:length: (page 464)
  Initializes an allocated NSIndexPath (page 459) object with an index path of a specific length.

Querying Index Paths

- getIndexes: (page 463)
  Provides a reference to the receiver's indexes.
- indexAtPosition: (page 463)
  Provides the index at a particular node in the receiver.
- indexPathByAddingIndex: (page 463)
  Provides an index path containing the indexes in the receiver and another index.
- indexPathByRemovingLastIndex (page 464)
  Provides an index path with the indexes in the receiver, excluding the last one.
- length (page 465)
  Provides the number of indexes in the receiver.

Comparing Index Paths

- compare: (page 462)
  Indicates the depth-first traversal order of the receiver and another index path.

Class Methods

indexPathWithIndex:

Creates an one-node index path.

```
+ (id)indexPathWithIndex:(NSUInteger)index
```

Parameters

index
  Index of the item in node 0 to point to.

Return Value

One-node index path with *index*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithIndex: (page 464)

Declared In

NSIndexPath.h

indexPathWithIndexes:length:

Creates an index path with one or more nodes.

```
+ (id)indexPathWithIndexes:(NSUInteger *)indexes length:(NSUInteger)length
```

Parameters

indexes

Array of indexes to make up the index path.

length

Number of nodes to include in the index path.

Return Value

Index path with *indexes* up to *length*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithIndexes:length: (page 464)

Declared In

NSIndexPath.h

Instance Methods

compare:

Indicates the depth-first traversal order of the receiver and another index path.

```
- (NSComparisonResult)compare:(NSIndexPath *)indexPath
```

Parameters

indexPath

Index path to compare.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

The depth-first traversal ordering of the receiver and *indexPath*.

- `NSOrderedAscending`: The receiver comes before *indexPath*.
- `NSOrderedDescending`: The receiver comes after *indexPath*.
- `NSOrderedSame`: The receiver and *indexPath* are the same index path.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSIndexPath.h

getIndexes:

Provides a reference to the receiver's indexes.

```
- (void)getIndexes:(NSUInteger *)indexes
```

Parameters

indexes

Pointer to an unsigned integer array. On return, the receiver indexes.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSIndexPath.h

indexAtPosition:

Provides the index at a particular node in the receiver.

```
- (NSUInteger)indexAtPosition:(NSUInteger)node
```

Parameters

node

Node with the desired index. Node numbering starts at zero.

Return Value

Index at *node*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSIndexPath.h

indexPathByAddingIndex:

Provides an index path containing the indexes in the receiver and another index.

```
- (NSIndexPath *)indexPathByAddingIndex:(NSUInteger)index
```

Parameters

index

Index to append to the receiver's indexes.

Return Value

New NSIndexPath (page 459) object containing the receiver's indexes and *index*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- indexPathByRemovingLastIndex (page 464)

Declared In
NSIndexPath.h

indexPathByRemovingLastIndex

Provides an index path with the indexes in the receiver, excluding the last one.

```
- (NSIndexPath *)indexPathByRemovingLastIndex
```

Return Value
New index path with the receiver's indexes, excluding the last one.

Availability
Available in iPhone OS 2.0 and later.

See Also
- indexPathByAddingIndex: (page 463)

Declared In
NSIndexPath.h

initWithIndex:

Initializes an allocated NSIndexPath (page 459) object with a one-node index path.

```
- (id)initWithIndex:(NSUInteger)index
```

Parameters
index
Index of the item in node 0 to point to.

Return Value
Initialized NSIndexPath (page 459) object representing a one-node index path with *index*.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ indexPathWithIndex: (page 461)

Declared In
NSIndexPath.h

initWithIndexes:length:

Initializes an allocated NSIndexPath (page 459) object with an index path of a specific length.

```
- (id)initWithIndexes:(NSUInteger *)indexes length:(NSUInteger)length
```

Parameters
indexes
Array of indexes to make up the index path.

length
Number of nodes to include in the index path.

Return Value
Initialized `NSIndexPath` (page 459) object with *indexes* up to *length*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ indexPathWithIndexes:length:` (page 462)

Declared In
`NSIndexPath.h`

length

Provides the number of indexes in the receiver.

```
- (NSUInteger)length
```

Return Value
Number of indexes in the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSIndexPath.h`

NSIndexSet Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSMutableCopying NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSIndexSet.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSIndexSet` class represents an immutable collection of unique unsigned integers, known as **indexes** because of the way they are used. This collection is referred to as a **index set**.

You use index sets in your code to store indexes into some other data structure. For example, given an `NSArray` object, you could use an index set to identify a subset of objects in that array.

Each index value can appear only once in the index set. This is an important concept to understand and is why you would not use index sets to store an arbitrary collection of integer values. To illustrate how this works, if you created an `NSIndexSet` object with the values `4`, `5`, `2`, and `5`, the resulting set would only have the values `4`, `5`, and `2` in it. Because index values are always maintained in sorted order, the actual order of the values when you created the set would be `2`, `4`, and then `5`.

In most cases, using an index set is more efficient than storing a collection of individual integers. Internally, the `NSIndexSet` class represents indexes using ranges. For maximum performance and efficiency, overlapping ranges in an index set are automatically coalesced—that is, ranges merge rather than overlap. Thus, the more contiguous the indexes in the set, the fewer ranges are required to specify those indexes.

The designated initializers of the NSIndexSet class are: `initWithIndexesInRange:` (page 477) and `initWithIndexSet:` (page 477).

You must not subclass the `NSIndexSet` class.

The mutable subclass of `NSIndexSet` is NSMutableIndexSet.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

NSMutableCopying
- `mutableCopyWithZone:` (page 1300)

Tasks

Creating Index Sets

`+ indexSet` (page 470)
Creates an empty index set.

`+ indexSetWithIndex:` (page 470)
Creates an index set with an index.

`+ indexSetWithIndexesInRange:` (page 470)
Creates an index set with an index range.

`- init` (page 476)
Initializes an allocated `NSIndexSet` (page 467) object.

`- initWithIndex:` (page 476)
Initializes an allocated `NSIndexSet` (page 467) object with an index.

- `initWithIndexesInRange:` (page 477)
  Initializes an allocated `NSIndexSet` (page 467) object with an index range.
- `initWithIndexSet:` (page 477)
  Initializes an allocated `NSIndexSet` (page 467) object with an index set.

Querying Index Sets

- `containsIndex:` (page 471)
  Indicates whether the receiver contains a specific index.
- `containsIndexes:` (page 471)
  Indicates whether the receiver contains a superset of the indexes in another index set.
- `containsIndexesInRange:` (page 472)
  Indicates whether the receiver contains the indexes represented by an index range.
- `intersectsIndexesInRange:` (page 478)
  Indicates whether the receiver contains any of the indexes in a range.
- `count` (page 472)
  Returns the number of indexes in the receiver.
- `countOfIndexesInRange:` (page 473)
  Returns the number of indexes in the receiver that are members of a given range.

Comparing Index Sets

- `isEqualToIndexSet:` (page 478)
  Indicates whether the indexes in the receiver are the same indeces contained in another index set.

Getting Indexes

- `firstIndex` (page 473)
  Returns either the first index in the receiver or the not-found indicator.
- `lastIndex` (page 479)
  Returns either the last index in the receiver or the not-found indicator.
- `indexLessThanIndex:` (page 475)
  Returns either the closest index in the receiver that is less than a specific index or the not-found indicator.
- `indexLessThanOrEqualToIndex:` (page 476)
  Returns either the closest index in the receiver that is less than or equal to a specific index or the not-found indicator.
- `indexGreaterThanOrEqualToIndex:` (page 475)
  Returns either the closest index in the receiver that is greater than or equal to a specific index or the not-found indicator.
- `indexGreaterThanIndex:` (page 474)
  Returns either the closest index in the receiver that is greater than a specific index or the not-found indicator.

- getIndexes:maxCount:inIndexRange: (page 473)
  The receiver fills an index buffer with the indexes contained both in the receiver and in an index range, returning the number of indexes copied.

Class Methods

indexSet

Creates an empty index set.

```
+ (id)indexSet
```

Return Value
NSIndexSet (page 467) object with no members.

Availability
Available in iPhone OS 2.0 and later.

See Also
- init (page 476)

Declared In
NSIndexSet.h

indexSetWithIndex:

Creates an index set with an index.

```
+ (id)indexSetWithIndex:(NSUInteger)index
```

Parameters
index
  An index.

Return Value
NSIndexSet (page 467) object containing *index*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithIndex: (page 476)

Declared In
NSIndexSet.h

indexSetWithIndexesInRange:

Creates an index set with an index range.

```
+ (id)indexSetWithIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange

An index range.

Return Value

NSIndexSet (page 467) object containing *indexRange*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- initWithIndexesInRange: (page 477)

Declared In

NSIndexSet.h

Instance Methods

containsIndex:

Indicates whether the receiver contains a specific index.

```
- (BOOL)containsIndex:(NSUInteger)index
```

Parameters

index

Index being inquired about.

Return Value

YES when the receiver contains *index*, NO otherwise.

Availability

Available in iPhone OS 2.0 and later.

See Also

- containsIndexes: (page 471)
- containsIndexesInRange: (page 472)

Declared In

NSIndexSet.h

containsIndexes:

Indicates whether the receiver contains a superset of the indexes in another index set.

```
- (BOOL)containsIndexes:(NSIndexSet *)indexSet
```

Parameters

indexSet

Index set being inquired about.

Return Value
YES when the receiver contains a superset of the indexes in *indexSet*, NO otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
- containsIndex: (page 471)
- containsIndexesInRange: (page 472)

Declared In
NSIndexSet.h

containsIndexesInRange:

Indicates whether the receiver contains the indexes represented by an index range.

```
- (BOOL)containsIndexesInRange:(NSRange)indexRange
```

Parameters
indexRange
The index range being inquired about.

Return Value
YES when the receiver contains the indexes in *indexRange*, NO otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
- containsIndex: (page 471)
- containsIndexes: (page 471)
- intersectsIndexesInRange: (page 478)

Declared In
NSIndexSet.h

count

Returns the number of indexes in the receiver.

```
- (NSUInteger)count
```

Return Value
Number of indexes in the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- countOfIndexesInRange: (page 473)

Declared In
NSIndexSet.h

countOfIndexesInRange:

Returns the number of indexes in the receiver that are members of a given range.

```
- (NSUInteger)countOfIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
Index range being inquired about.

Return Value
Number of indexes in the receiver that are members of *indexRange*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- count (page 472)

Declared In
NSIndexSet.h

firstIndex

Returns either the first index in the receiver or the not-found indicator.

```
- (NSUInteger)firstIndex
```

Return Value
First index in the receiver or NSNotFound (page 1418) when the receiver is empty.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lastIndex (page 479)

Declared In
NSIndexSet.h

getIndexes:maxCount:inIndexRange:

The receiver fills an index buffer with the indexes contained both in the receiver and in an index range, returning the number of indexes copied.

```
- (NSUInteger)getIndexes:(NSUInteger *)indexBuffer maxCount:(NSUInteger)bufferSize
    inIndexRange:(NSRangePointer)indexRangePointer
```

Parameters

indexBuffer

Index buffer to fill.

bufferSize

Maximum size of *indexBuffer*.

indexRange

Index range to compare with indexes in the receiver; `nil` represents all the indexes in the receiver. Indexes in the index range and in the receiver are copied to *indexBuffer*. On output, the range of indexes not copied to *indexBuffer*.

Return Value

Number of indexes placed in *indexBuffer*.

Discussion

You are responsible for allocating the memory required for *indexBuffer* and for releasing it later.

Suppose you have an index set with contiguous indexes from 1 to 100. If you use this method to request a range of `(1, 100)`—which represents the set of indexes 1 through 100—and specify a buffer size of `20`, this method returns 20 indexes—1 through 20—in *indexBuffer* and sets *indexRange* to `(21, 80)`—which represents the indexes 21 through 100.

Use this method to retrieve entries quickly and efficiently from an index set. You can call this method repeatedly to retrieve blocks of index values and then process them. When doing so, use the return value and *indexRange* to determine when you have finished processing the desired indexes. When the return value is less than *bufferSize*, you have reached the end of the range.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSIndexSet.h`

indexGreaterThanIndex:

Returns either the closest index in the receiver that is greater than a specific index or the not-found indicator.

```
- (NSUInteger)indexGreaterThanIndex:(NSUInteger)index
```

Parameters

index

Index being inquired about.

Return Value

Closest index in the receiver greater than *index*; `NSNotFound` (page 1418) when the receiver contains no qualifying index.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `indexLessThanIndex:` (page 475)
- `indexGreaterThanOrEqualToIndex:` (page 475)
- `indexLessThanOrEqualToIndex:` (page 476)

Declared In
NSIndexSet.h

indexGreaterThanOrEqualToIndex:

Returns either the closest index in the receiver that is greater than or equal to a specific index or the not-found indicator.

```
- (NSUInteger)indexGreaterThanOrEqualToIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver greater than or equal to *index*; NSNotFound (page 1418) when the receiver contains no qualifying index.

Availability
Available in iPhone OS 2.0 and later.

See Also
- indexGreaterThanIndex: (page 474)
- indexLessThanIndex: (page 475)
- indexLessThanOrEqualToIndex: (page 476)

Declared In
NSIndexSet.h

indexLessThanIndex:

Returns either the closest index in the receiver that is less than a specific index or the not-found indicator.

```
- (NSUInteger)indexLessThanIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver less than *index*; NSNotFound (page 1418) when the receiver contains no qualifying index.

Availability
Available in iPhone OS 2.0 and later.

See Also
- indexGreaterThanIndex: (page 474)
- indexGreaterThanOrEqualToIndex: (page 475)
- indexLessThanOrEqualToIndex: (page 476)

Declared In
NSIndexSet.h

indexLessThanOrEqualToIndex:

Returns either the closest index in the receiver that is less than or equal to a specific index or the not-found indicator.

```
- (NSUInteger)indexLessThanOrEqualToIndex:(NSUInteger)index
```

Parameters

index
Index being inquired about.

Return Value
Closest index in the receiver less than or equal to *index*; NSNotFound (page 1418) when the receiver contains no qualifying index.

Availability
Available in iPhone OS 2.0 and later.

See Also
- indexGreaterThanIndex: (page 474)
- indexLessThanIndex: (page 475)
- indexGreaterThanOrEqualToIndex: (page 475)

Declared In
NSIndexSet.h

init

Initializes an allocated NSIndexSet (page 467) object.

```
- (id)init
```

Return Value
Initialized, empty NSIndexSet (page 467) object.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ indexSet (page 470)

Declared In
NSIndexSet.h

initWithIndex:

Initializes an allocated NSIndexSet (page 467) object with an index.

```
- (id)initWithIndex:(NSUInteger)index
```

Parameters

index

An index.

Return Value

Initialized `NSIndexSet` (page 467) object with *index*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ indexSetWithIndex:` (page 470)

Declared In

`NSIndexSet.h`

initWithIndexesInRange:

Initializes an allocated `NSIndexSet` (page 467) object with an index range.

```
- (id)initWithIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange

An index range. Must include only indexes representable as unsigned integers.

Return Value

Initialized `NSIndexSet` (page 467) object with *indexRange*.

Discussion

This method raises an `NSRangeException` when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

This method is a designated initializer for `NSIndexSet` (page 467).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ indexSetWithIndexesInRange:` (page 470)

Declared In

`NSIndexSet.h`

initWithIndexSet:

Initializes an allocated `NSIndexSet` (page 467) object with an index set.

```
- (id)initWithIndexSet:(NSIndexSet *)indexSet
```

Parameters

indexSet

An index set.

Return Value
Initialized `NSIndexSet` (page 467) object with *`indexSet`*.

Discussion
This method is a designated initializer for `NSIndexSet` (page 467).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSIndexSet.h`

intersectsIndexesInRange:

Indicates whether the receiver contains any of the indexes in a range.

```
- (BOOL)intersectsIndexesInRange:(NSRange)indexRange
```

Parameters
`indexRange`
Index range being inquired about.

Return Value
`YES` when the receiver contains one or more of the indexes in *`indexRange`*, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- containsIndexesInRange:` (page 472)

Declared In
`NSIndexSet.h`

isEqualToIndexSet:

Indicates whether the indexes in the receiver are the same indeces contained in another index set.

```
- (BOOL)isEqualToIndexSet:(NSIndexSet *)indexSet
```

Parameters
`indexSet`
Index set being inquired about.

Return Value
`YES` when the indexes in the receiver are the same indexes *`indexSet`* contains, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSIndexSet.h`

lastIndex

Returns either the last index in the receiver or the not-found indicator.

```
- (NSUInteger)lastIndex
```

Return Value

Last index in the receiver or `NSNotFound` (page 1418) when the receiver is empty.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- firstIndex` (page 473)

Declared In

`NSIndexSet.h`

NSInputStream Class Reference

Inherits from:	NSStream : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSStream.h
Companion guide:	Stream Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSInputStream` is a subclass of `NSStream` that provides read-only stream functionality.

Subclassing Notes

`NSInputStream` is a concrete subclass of `NSStream` that gives you standard read-only access to stream data. Although `NSInputStream` is probably sufficient for most situations requiring access to stream data, you can create a subclass of `NSInputStream` if you want more specialized behavior (for example, you want to record statistics on the data in a stream).

Methods to Override

To create a subclass of `NSInputStream` you may have to implement initializers for the type of stream data supported and suitably reimplement existing initializers. You must also provide complete implementations of the following methods:

- `read:maxLength:` (page 485)

  From the current read index, take up to the number of bytes specified in the second parameter from the stream and place them in the client-supplied buffer (first parameter). The buffer must be of the size specified by the second parameter. Return the actual number of bytes placed in the buffer; if there is nothing left in the stream, return `0`. Reset the index into the stream for the next read operation.

- `getBuffer:length:` (page 484)

  Return in 0(1) a pointer to the subclass-allocated buffer (first parameter). Return by reference in the second parameter the number of bytes actually put into the buffer. The buffer's contents are valid only until the next stream operation. Return `NO` if you cannot access data in the buffer; otherwise, return `YES`. If this method is not appropriate for your type of stream, you may return `NO`.

- `hasBytesAvailable` (page 484)

  Return `YES` if there is more data to read in the stream, `NO` if there is not. If you want to be semantically compatible with `NSInputStream`, return `YES` if a read must be attempted to determine if bytes are available.

Tasks

Creating Streams

`+ inputStreamWithData:` (page 483)
Creates and returns an initialized `NSInputStream` object for reading from a given `NSData` object.

`+ inputStreamWithFileAtPath:` (page 483)
Creates and returns an initialized `NSInputStream` object that reads data from the file at a given path.

`- initWithData:` (page 485)
Initializes and returns an `NSInputStream` object for reading from a given `NSData` object.

`- initWithFileAtPath:` (page 485)
Initializes and returns an `NSInputStream` object that reads data from the file at a given path.

Using Streams

`- read:maxLength:` (page 485)
Reads up to a given number of bytes into a given buffer, and returns the actual number of bytes read.

- getBuffer:length: (page 484)
  Returns by reference a pointer to a read buffer and, by reference, the number of bytes available, and returns a Boolean value that indicates whether the buffer is available.
- hasBytesAvailable (page 484)
  Returns a Boolean value that indicates whether the receiver has bytes available to read.

Class Methods

inputStreamWithData:

Creates and returns an initialized NSInputStream object for reading from a given NSData object.

```
+ (id)inputStreamWithData:(NSData *)data
```

Parameters

data
The data object from which to read. The contents of *data* are copied.

Return Value

An initialized NSInputStream object for reading from *data*. If *data* is not an NSData object, this method returns nil.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ inputStreamWithFileAtPath: (page 483)
- initWithData: (page 485)

Declared In

NSStream.h

inputStreamWithFileAtPath:

Creates and returns an initialized NSInputStream object that reads data from the file at a given path.

```
+ (id)inputStreamWithFileAtPath:(NSString *)path
```

Parameters

path
The path to the file.

Return Value

An initialized NSInputStream object that reads data from the file at *path*. If the file specified by *path* doesn't exist or is unreadable, returns nil.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ inputStreamWithData: (page 483)

- initWithFileAtPath: (page 485)

Declared In
NSStream.h

Instance Methods

getBuffer:length:

Returns by reference a pointer to a read buffer and, by reference, the number of bytes available, and returns a Boolean value that indicates whether the buffer is available.

```
- (BOOL)getBuffer:(uint8_t **)buffer length:(NSUInteger *)len
```

Parameters

buffer

Upon return, contains a pointer to a read buffer. The buffer is only valid until the next stream operation is performed.

len

Upon return, contains the number of bytes available.

Return Value
YES if the buffer is available, otherwise NO.

Subclasses of NSInputStream may return NO if this operation is not appropriate for the stream type.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSStream.h

hasBytesAvailable

Returns a Boolean value that indicates whether the receiver has bytes available to read.

```
- (BOOL)hasBytesAvailable
```

Return Value
YES if the receiver has bytes available to read, otherwise NO. May also return YES if a read must be attempted in order to determine the availability of bytes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSStream.h

initWithData:

Initializes and returns an `NSInputStream` object for reading from a given `NSData` object.

```
- (id)initWithData:(NSData *)data
```

Parameters

data

The data object from which to read. The contents of *data* are copied.

Return Value

An initialized `NSInputStream` object for reading from *data*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithFileAtPath:` (page 485)
+ `inputStreamWithData:` (page 483)

Declared In

`NSStream.h`

initWithFileAtPath:

Initializes and returns an `NSInputStream` object that reads data from the file at a given path.

```
- (id)initWithFileAtPath:(NSString *)path
```

Parameters

path

The path to the file.

Return Value

An initialized `NSInputStream` object that reads data from the file at *path*. If the file specified by *path* doesn't exist or is unreadable, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithData:` (page 485)
+ `inputStreamWithFileAtPath:` (page 483)

Declared In

`NSStream.h`

read:maxLength:

Reads up to a given number of bytes into a given buffer, and returns the actual number of bytes read.

```
- (NSInteger)read:(uint8_t *)buffer maxLength:(NSUInteger)len
```

Parameters

buffer

A data buffer. The buffer must be large enough to contain the number of bytes specified by *len*.

len

The maximum number of bytes to read.

Return Value

The actual number of bytes read.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSStream.h

NSInvocation Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSInvocation.h
Companion guide:	Distributed Objects

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSInvocation` is an Objective-C message rendered static, that is, it is an action turned into an object. `NSInvocation` objects are used to store and forward messages between objects and between applications, primarily by `NSTimer` objects and the distributed objects system.

An `NSInvocation` object contains all the elements of an Objective-C message: a target, a selector, arguments, and the return value. Each of these elements can be set directly, and the return value is set automatically when the `NSInvocation` object is dispatched.

An `NSInvocation` object can be repeatedly dispatched to different targets; its arguments can be modified between dispatch for varying results; even its selector can be changed to another with the same method signature (argument and return types). This flexibility makes `NSInvocation` useful for

repeating messages with many arguments and variations; rather than retyping a slightly different expression for each message, you modify the `NSInvocation` object as needed each time before dispatching it to a new target.

`NSInvocation` does not support invocations of methods with either variable numbers of arguments or `union` arguments. You should use the `invocationWithMethodSignature:` (page 489) class method to create `NSInvocation` objects; you should not create these objects using `alloc` (page 783) and `init` (page 803).

This class does not retain the arguments for the contained invocation by default. If those objects might disappear between the time you create your instance of `NSInvocation` and the time you use it, you should explicitly retain the objects yourself or invoke the `retainArguments` method to have the invocation object retain them itself.

Note: `NSInvocation` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSInvocation` does not support archiving.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

Tasks

Creating NSInvocation Objects

`+ invocationWithMethodSignature:` (page 489)
Returns an `NSInvocation` object able to construct messages using a given method signature.

Configuring an Invocation Object

`- setSelector:` (page 494)
Sets the receiver's selector.

`- selector` (page 493)
Returns the receiver's selector, or 0 if it hasn't been set.

`- setTarget:` (page 495)
Sets the receiver's targe.

`- target` (page 495)
Returns the receiver's target, or `nil` if the receiver has no target.

`- setArgument:atIndex:` (page 493)
Sets an argument of the receiver.

- `getArgument:atIndex:` (page 490)
  Returns by indirection the receiver's argument at a specified index.
- `argumentsRetained` (page 490)
  Returns `YES` if the receiver has retained its arguments, `NO` otherwise.
- `retainArguments` (page 493)
  If the receiver hasn't already done so, retains the target and all object arguments of the receiver and copies all of its C-string arguments.
- `setReturnValue:` (page 494)
  Sets the receiver's return value.
- `getReturnValue:` (page 491)
  Gets the receiver's return value.

Dispatching an Invocation

- `invoke` (page 491)
  Sends the receiver's message (with arguments) to its target and sets the return value.
- `invokeWithTarget:` (page 492)
  Sets the receiver's target, sends the receiver's message (with arguments) to that target, and sets the return value.

Getting the Method Signature

- `methodSignature` (page 492)
  Returns the receiver's method signature.

Class Methods

invocationWithMethodSignature:

Returns an `NSInvocation` object able to construct messages using a given method signature.

```
+ (NSInvocation *)invocationWithMethodSignature:(NSMethodSignature *)signature
```

Parameters

signature
  An object encapsulating a method signature.

Discussion

The new object must have its selector set with `setSelector:` (page 494) and its arguments set with `setArgument:atIndex:` (page 493) before it can be invoked. Do not use the `alloc` (page 783)/`init` (page 803) approach to create `NSInvocation` objects.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSInvocation.h

Instance Methods

argumentsRetained

Returns YES if the receiver has retained its arguments, NO otherwise.

```
- (BOOL)argumentsRetained
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- retainArguments (page 493)

Declared In
NSInvocation.h

getArgument:atIndex:

Returns by indirection the receiver's argument at a specified index.

```
- (void)getArgument:(void *)buffer atIndex:(NSInteger)index
```

Parameters

buffer

An untyped buffer to hold the returned argument. See the discussion below relating to argument values that are objects.

index

An integer specifying the index of the argument to get.

Indices 0 and 1 indicate the hidden arguments *self* and *_cmd*, respectively; these values can be retrieved directly with the target and selector methods. Use indices 2 and greater for the arguments normally passed in a message.

Discussion
This method copies the argument stored at *index* into the storage pointed to by *buffer*. The size of *buffer* must be large enough to accommodate the argument value.

When the argument value is an object, pass a pointer to the variable (or memory) into which the object should be placed:

```
NSArray *anArray;
[invocation getArgument:&anArray atIndex:3];
```

This method raises NSInvalidArgumentException if *index* is greater than the actual number of arguments for the selector.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setArgument:atIndex:` (page 493)
- `numberOfArguments` (page 564) (NSMethodSignature)

Declared In
`NSInvocation.h`

getReturnValue:
Gets the receiver's return value.

```
- (void)getReturnValue:(void *)buffer
```

Parameters
buffer

An untyped buffer into which the receiver copies its return value. It should be large enough to accommodate the value. See the discussion below for more information about *buffer*.

Discussion
Use the `NSMethodSignature` method `methodReturnLength` (page 564) to determine the size needed for *buffer*:

```
unsigned int length = [[myInvocation methodSignature] methodReturnLength];
buffer = (void *)malloc(length);
[invocation getReturnValue:buffer];
```

When the return value is an object, pass a pointer to the variable (or memory) into which the object should be placed:

```
id anObject;
NSArray *anArray;
[invocation1 getReturnValue:&anObject];
[invocation2 getReturnValue:&anArray];
```

If the `NSInvocation` object has never been invoked, the result of this method is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setReturnValue:` (page 494)
- `methodReturnType` (page 564) (NSMethodSignature)

Declared In
`NSInvocation.h`

invoke
Sends the receiver's message (with arguments) to its target and sets the return value.

```
- (void)invoke
```

Discussion
You must set the receiver's target, selector, and argument values before calling this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `getReturnValue:` (page 491)
- `setSelector:` (page 494)
- `setTarget:` (page 495)
- `setArgument:atIndex:` (page 493)

Declared In
`NSInvocation.h`

invokeWithTarget:

Sets the receiver's target, sends the receiver's message (with arguments) to that target, and sets the return value.

```
- (void)invokeWithTarget:(id)anObject
```

Parameters
anObject
The object to set as the receiver's target.

Discussion
You must set the receiver's selector and argument values before calling this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `getReturnValue:` (page 491)
- `invoke` (page 491)
- `setSelector:` (page 494)
- `setTarget:` (page 495)
- `setArgument:atIndex:` (page 493)

Declared In
`NSInvocation.h`

methodSignature

Returns the receiver's method signature.

```
- (NSMethodSignature *)methodSignature
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSInvocation.h`

retainArguments

If the receiver hasn't already done so, retains the target and all object arguments of the receiver and copies all of its C-string arguments.

```
- (void)retainArguments
```

Discussion

Before this method is invoked, `argumentsRetained` (page 490) returns `NO`; after, it returns `YES`.

For efficiency, newly created NSInvocations don't retain or copy their arguments, nor do they retain their targets or copy C strings. You should instruct an NSInvocation to retain its arguments if you intend to cache it, since the arguments may otherwise be released before the NSInvocation is invoked. NSTimers always instruct their NSInvocations to retain their arguments, for example, because there's usually a delay before an NSTimer fires.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSInvocation.h`

selector

Returns the receiver's selector, or 0 if it hasn't been set.

```
- (SEL)selector
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setSelector:` (page 494)

Declared In

`NSInvocation.h`

setArgument:atIndex:

Sets an argument of the receiver.

```
- (void)setArgument:(void *)buffer atIndex:(NSInteger)index
```

Parameters

buffer

An untyped buffer containing an argument to be assigned to the receiver. See the discussion below relating to argument values that are objects.

index

An integer specifying the index of the argument.

Indices 0 and 1 indicate the hidden arguments *self* and *_cmd*, respectively; you should set these values directly with the `setTarget:` (page 495) and `setSelector:` (page 494) methods. Use indices 2 and greater for the arguments normally passed in a message.

Discussion

This method copies the contents of *buffer* as the argument at *index*. The number of bytes copied is determined by the argument size.

When the argument value is an object, pass a pointer to the variable (or memory) from which the object should be copied:

```
NSArray *anArray;
[invocation setArgument:&anArray atIndex:3];
```

This method raises `NSInvalidArgumentException` if the value of *index* is greater than the actual number of arguments for the selector.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `getArgument:atIndex:` (page 490)
- `numberOfArguments` (page 564) (NSMethodSignature)

Declared In

`NSInvocation.h`

setReturnValue:

Sets the receiver's return value.

```
- (void)setReturnValue:(void *)buffer
```

Parameters

buffer

An untyped buffer whose contents are copied as the receiver's return value.

Discussion

This value is normally set when you send an `invoke` (page 491) or `invokeWithTarget:` (page 492) message.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `getReturnValue:` (page 491)
- `methodReturnLength` (page 564) (NSMethodSignature)
- `methodReturnType` (page 564) (NSMethodSignature)

Declared In

`NSInvocation.h`

setSelector:

Sets the receiver's selector.

```
- (void)setSelector:(SEL)selector
```

Parameters

selector

The selector to assign to the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- selector (page 493)

Declared In

NSInvocation.h

setTarget:

Sets the receiver's targe.

```
- (void)setTarget:(id)anObject
```

Parameters

anObject

The object to assign to the receiver as target. The target is the receiver of the message sent by invoke (page 491).

Discussion

Availability

Available in iPhone OS 2.0 and later.

See Also

- target (page 495)
- invokeWithTarget: (page 492)

Declared In

NSInvocation.h

target

Returns the receiver's target, or nil if the receiver has no target.

```
- (id)target
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- setTarget: (page 495)

Declared In

NSInvocation.h

NSInvocationOperation Class Reference

Inherits from:	NSOperation : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSOperation.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSInvocationOperation` class is a concrete subclass of `NSOperation` that manages the execution of a single encapsulated task specified as an invocation. You can use this class to initiate an operation that consists of invoking a selector on a specified object. This class implements a non-concurrent operation.

For more information on concurrent versus non-concurrent operations, see *NSOperation Class Reference*.

Tasks

Initialization

- `initWithTarget:selector:object:` (page 498)
  Returns an `NSInvocationOperation` object initialized with the specified target and selector.
- `initWithInvocation:` (page 498)
  Returns an `NSInvocationOperation` object initialized with the specified invocation object.

Getting Attributes

- `invocation` (page 499)
  Returns the receiver's invocation object.
- `result` (page 499)
  Returns the result of the invocation or method.

Instance Methods

initWithInvocation:

Returns an `NSInvocationOperation` object initialized with the specified invocation object.

```
- (id)initWithInvocation:(NSInvocation *)inv
```

Parameters

inv
  The invocation object identifying the target object, selector, and parameter objects.

Return Value

An initialized `NSInvocationOperation` object or `nil` if the object could not be initialized.

Discussion

This method is the designated initializer. The receiver tells the invocation object to retain its arguments.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSOperation.h`

initWithTarget:selector:object:

Returns an `NSInvocationOperation` object initialized with the specified target and selector.

```
- (id)initWithTarget:(id)target selector:(SEL)sel object:(id)arg
```

Parameters

target

The object defining the specified selector.

sel

The selector to invoke when running the operation. The selector may take 0 or 1 parameters. If it accepts a parameter, the type of that parameter should be `id`.

arg

The parameter object to pass to the selector. If the selector does not take an argument, specify `nil`.

Return Value

An initialized `NSInvocationOperation` object or `nil` if the target object does not implement the specified selector.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSOperation.h`

invocation

Returns the receiver's invocation object.

```
- (NSInvocation *)invocation
```

Return Value

The invocation object identifying the target object, selector, and parameters to use to execute the operation's task.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithTarget:selector:object:` (page 498)

`- initWithInvocation:` (page 498)

Declared In

`NSOperation.h`

result

Returns the result of the invocation or method.

```
- (id)result
```

Return Value

The object returned by the method or an `NSValue` object containing the return value if it is not an object. If the method or invocation is not finished executing, this method returns `nil`.

Discussion
If an exception was raised during the execution of the method or invocation, this method raises that exception again. If the operation was cancelled or the invocation or method has a `void` return type, calling this method raises an exception; see "Result Exceptions" (page 500).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSOperation.h`

Constants

Result Exceptions

Names of exceptions raised by `NSInvocationOperation` if there is an error when calling the `result` (page 499) method.

```
extern NSString * const NSInvocationOperationVoidResultException;
extern NSString * const NSInvocationOperationCancelledException;
```

Constants

`NSInvocationOperationVoidResultException`
The name of the exception raised if the `result` method is called for an invocation method with a `void` return type.

Available in iPhone OS 2.0 and later.

Declared in `NSOperation.h`

`NSInvocationOperationCancelledException`
The name of the exception raised if the `result` method is called after the operation was cancelled.

Available in iPhone OS 2.0 and later.

Declared in `NSOperation.h`

Declared In
`NSOperation.h`

NSKeyedArchiver Class Reference

Inherits from:	NSCoder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSKeyedArchiver.h
Companion guide:	Archives and Serializations Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSKeyedArchiver`, a concrete subclass of `NSCoder`, provides a way to encode objects (and scalar values) into an architecture-independent format that can be stored in a file. When you archive a set of objects, the class information and instance variables for each object are written to the archive. `NSKeyedArchiver`'s companion class, `NSKeyedUnarchiver`, decodes the data in an archive and creates a set of objects equivalent to the original set.

A keyed archive differs from a non-keyed archive in that all the objects and values encoded into the archive are given names, or keys. When decoding a non-keyed archive, values have to be decoded in the same order in which they were encoded. When decoding a keyed archive, because values are requested by name, values can be decoded out of sequence or not at all. Keyed archives, therefore, provide better support for forward and backward compatibility.

The keys given to encoded values must be unique only within the scope of the current object being encoded. A keyed archive is hierarchical, so the keys used by object A to encode its instance variables do not conflict with the keys used by object B, even if A and B are instances of the same class. Within a single object, however, the keys used by a subclass can conflict with keys used in its superclasses.

An `NSArchiver` object can write the archive data to a file or to a mutable-data object (an instance of `NSMutableData`) that you provide.

Tasks

Initializing an NSKeyedArchiver Object

`- initForWritingWithMutableData:` (page 510)
Returns the receiver, initialized for encoding an archive into a given a mutable-data object.

Archiving Data

`+ archivedDataWithRootObject:` (page 504)
Returns an `NSData` object containing the encoded form of the object graph whose root object is given.

`+ archiveRootObject:toFile:` (page 504)
Archives an object graph rooted at a given object by encoding it into a data object then atomically writes the resulting data object to a file at a given path, and returns a Boolean value that indicates whether the operation was successful.

`- finishEncoding` (page 510)
Instructs the receiver to construct the final data stream.

`- outputFormat` (page 511)
Returns the format in which the receiver encodes its data.

`- setOutputFormat:` (page 512)
Sets the format in which the receiver encodes its data.

Encoding Data and Objects

`- archiver:didEncodeObject:` (page 513) *delegate method*
Informs the delegate that a given object has been encoded.

`- archiverDidFinish:` (page 514) *delegate method*
Notifies the delegate that encoding has finished.

`- archiver:willEncodeObject:` (page 513) *delegate method*
Informs the delegate that *`object`* is about to be encoded.

`- archiverWillFinish:` (page 514) *delegate method*
Notifies the delegate that encoding is about to finish.

`- archiver:willReplaceObject:withObject:` (page 514) *delegate method*
Informs the delegate that one given object is being substituted for another given object.

- `encodeBool:forKey:` (page 506)
  Encodes a given Boolean value and associates it with a given key.
- `encodeBytes:length:forKey:` (page 507)
  Encodes a given number of bytes from a given C array of bytes and associates them with the a given key.
- `encodeConditionalObject:forKey:` (page 507)
  Encodes a reference to a given object and associates it with a given key only if it has been unconditionally encoded elsewhere in the archive with `encodeObject:forKey:` (page 510).
- `encodeDouble:forKey:` (page 508)
  Encodes a given `double` value and associates it with a given key.
- `encodeFloat:forKey:` (page 508)
  Encodes a given `float` value and associates it with a given key.
- `encodeInt:forKey:` (page 509)
  Encodes a given `int` value and associates it with a given key.
- `encodeInt32:forKey:` (page 509)
  Encodes a given 32-bit integer value and associates it with a given key.
- `encodeInt64:forKey:` (page 509)
  Encodes a given 64-bit integer value and associates it with a given key.
- `encodeObject:forKey:` (page 510)
  Encodes a given object and associates it with a given key.

Managing Delegates

- `delegate` (page 506)
  Returns the receiver's delegate.
- `setDelegate:` (page 512)
  Sets the delegate for the receiver.

Managing Classes and Class Names

+ `setClassName:forClass:` (page 505)
  Adds a class translation mapping to `NSKeyedArchiver` whereby instances of of a given class are encoded with a given class name instead of their real class names.
+ `classNameForClass:` (page 504)
  Returns the class name with which `NSKeyedArchiver` encodes instances of a given class.
- `setClassName:forClass:` (page 511)
  Adds a class translation mapping to the receiver whereby instances of of a given class are encoded with a given class name instead of their real class names.
- `classNameForClass:` (page 506)
  Returns the class name with which the receiver encodes instances of a given class.

Class Methods

archivedDataWithRootObject:

Returns an `NSData` object containing the encoded form of the object graph whose root object is given.

```
+ (NSData *)archivedDataWithRootObject:(id)rootObject
```

Parameters

rootObject

The root of the object graph to archive.

Return Value

An `NSData` object containing the encoded form of the object graph whose root object is *rootObject*. The format of the archive is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyedArchiver.h`

archiveRootObject:toFile:

Archives an object graph rooted at a given object by encoding it into a data object then atomically writes the resulting data object to a file at a given path, and returns a Boolean value that indicates whether the operation was successful.

```
+ (BOOL)archiveRootObject:(id)rootObject toFile:(NSString *)path
```

Parameters

rootObject

The root of the object graph to archive.

path

The path of the file in which to write the archive.

Return Value

`YES` if the operation was successful, otherwise `NO`.

Discussion

The format of the archive is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyedArchiver.h`

classNameForClass:

Returns the class name with which `NSKeyedArchiver` encodes instances of a given class.

```
+ (NSString *)classNameForClass:(Class)cls
```

Parameters

cls

The class for which to determine the translation mapping.

Return Value

The class name with which `NSKeyedArchiver` encodes instances of *cls*. Returns `nil` if `NSKeyedArchiver` does not have a translation mapping for *cls*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ setClassName:forClass:` (page 505)

`- classNameForClass:` (page 506)

Declared In

`NSKeyedArchiver.h`

setClassName:forClass:

Adds a class translation mapping to `NSKeyedArchiver` whereby instances of of a given class are encoded with a given class name instead of their real class names.

```
+ (void)setClassName:(NSString *)codedName forClass:(Class)cls
```

Parameters

codedName

The name of the class that `NSKeyedArchiver` uses in place of `cls`.

cls

The class for which to set up a translation mapping.

Discussion

When encoding, the class's translation mapping is used only if no translation is found first in an instance's separate translation map.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ classNameForClass:` (page 504)

`- setClassName:forClass:` (page 511)

Declared In

`NSKeyedArchiver.h`

Instance Methods

classNameForClass:

Returns the class name with which the receiver encodes instances of a given class.

```
- (NSString *)classNameForClass:(Class)cls
```

Parameters

cls

The class for which to determine the translation mapping.

Return Value

The class name with which the receiver encodes instances of *cls*. Returns `nil` if the receiver does not have a translation mapping for *cls*. The class's separate translation map is not searched.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setClassName:forClass:` (page 511)
+ `classNameForClass:` (page 504)

Declared In

`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value

The receiver's delegate.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setDelegate:` (page 512)

Declared In

`NSKeyedArchiver.h`

encodeBool:forKey:

Encodes a given Boolean value and associates it with a given key.

```
- (void)encodeBool:(BOOL)boolv forKey:(NSString *)key
```

Parameters

boolv

The value to encode.

key

The key with which to associate *boolv*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`decodeBoolForKey:` (page 522) (NSKeyedUnarchiver)

Declared In

`NSKeyedArchiver.h`

encodeBytes:length:forKey:

Encodes a given number of bytes from a given C array of bytes and associates them with the a given key.

```
- (void)encodeBytes:(const uint8_t *)bytesp length:(NSUInteger)lenv forKey:(NSString *)key
```

Parameters

bytesp

A C array of bytes to encode.

lenv

The number of bytes from *bytesp* to encode.

key

The key with which to associate the encoded value.

Availability

Available in iPhone OS 2.0 and later.

See Also

`decodeBytesForKey:returnedLength:` (page 523) (NSKeyedUnarchiver)

Declared In

`NSKeyedArchiver.h`

encodeConditionalObject:forKey:

Encodes a reference to a given object and associates it with a given key only if it has been unconditionally encoded elsewhere in the archive with `encodeObject:forKey:` (page 510).

```
- (void)encodeConditionalObject:(id)objv forKey:(NSString *)key
```

Parameters

objv

The object to encode.

key
The key with which to associate the encoded value.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

encodeDouble:forKey:

Encodes a given `double` value and associates it with a given key.

```
- (void)encodeDouble:(double)realv forKey:(NSString *)key
```

Parameters

realv
The value to encode.

key
The key with which to associate *realv*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`decodeDoubleForKey:` (page 523) (NSKeyedUnarchiver)
`decodeFloatForKey:` (page 524) (NSKeyedUnarchiver)

Declared In
NSKeyedArchiver.h

encodeFloat:forKey:

Encodes a given `float` value and associates it with a given key.

```
- (void)encodeFloat:(float)realv forKey:(NSString *)key
```

Parameters

realv
The value to encode.

key
The key with which to associate *realv*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`decodeFloatForKey:` (page 524) (NSKeyedUnarchiver)
`decodeDoubleForKey:` (page 523) (NSKeyedUnarchiver)

Declared In
NSKeyedArchiver.h

encodeInt32:forKey:

Encodes a given 32-bit integer value and associates it with a given key.

```
- (void)encodeInt32:(int32_t)intv forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`decodeInt32ForKey:` (page 524) (NSKeyedUnarchiver)

Declared In

`NSKeyedArchiver.h`

encodeInt64:forKey:

Encodes a given 64-bit integer value and associates it with a given key.

```
- (void)encodeInt64:(int64_t)intv forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`decodeInt64ForKey:` (page 525) (NSKeyedUnarchiver)

Declared In

`NSKeyedArchiver.h`

encodeInt:forKey:

Encodes a given `int` value and associates it with a given key.

```
- (void)encodeInt:(int)intv forKey:(NSString *)key
```

Parameters

intv

The value to encode.

key

The key with which to associate *intv*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`decodeIntForKey:` (page 525) (NSKeyedUnarchiver)

Declared In
`NSKeyedArchiver.h`

encodeObject:forKey:

Encodes a given object and associates it with a given key.

```
- (void)encodeObject:(id)objv forKey:(NSString *)key
```

Parameters

objv
The value to encode.

key
The key with which to associate *objv*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`decodeObjectForKey:` (page 526) (NSKeyedUnarchiver)

Declared In
`NSKeyedArchiver.h`

finishEncoding

Instructs the receiver to construct the final data stream.

```
- (void)finishEncoding
```

Discussion
No more values can be encoded after this method is called. You must call this method when finished.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initForWritingWithMutableData:` (page 510)

Declared In
`NSKeyedArchiver.h`

initForWritingWithMutableData:

Returns the receiver, initialized for encoding an archive into a given a mutable-data object.

```
- (id)initForWritingWithMutableData:(NSMutableData *)data
```

Parameters

data

The mutable-data object into which the archive is written.

Return Value

The receiver, initialized for encoding an archive into *data*.

Discussion

When you finish encoding data, you must invoke `finishEncoding` (page 510) at which point *data* is filled. The format of the receiver is `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyedArchiver.h`

outputFormat

Returns the format in which the receiver encodes its data.

```
- (NSPropertyListFormat)outputFormat
```

Return Value

The format in which the receiver encodes its data. The available formats are `NSPropertyListXMLFormat_v1_0` and `NSPropertyListBinaryFormat_v1_0`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setOutputFormat:` (page 512)

Declared In

`NSKeyedArchiver.h`

setClassName:forClass:

Adds a class translation mapping to the receiver whereby instances of of a given class are encoded with a given class name instead of their real class names.

```
- (void)setClassName:(NSString *)codedName forClass:(Class)cls
```

Parameters

codedName

The name of the class that the receiver uses uses in place of `cls`.

cls

The class for which to set up a translation mapping.

Discussion

When encoding, the receiver's translation map overrides any translation that may also be present in the class's map.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `classNameForClass:` (page 506)
+ `setClassName:forClass:` (page 505)

Declared In
`NSKeyedArchiver.h`

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id)delegate
```

Parameters
delegate
The delegate for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `delegate` (page 506)

Declared In
`NSKeyedArchiver.h`

setOutputFormat:

Sets the format in which the receiver encodes its data.

```
- (void)setOutputFormat:(NSPropertyListFormat)format
```

Parameters
format
The format in which the receiver encodes its data. *format* can be `NSPropertyListXMLFormat_v1_0` or `NSPropertyListBinaryFormat_v1_0`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `outputFormat` (page 511)

Declared In
`NSKeyedArchiver.h`

Delegate Methods

archiver:didEncodeObject:

Informs the delegate that a given object has been encoded.

```
- (void)archiver:(NSKeyedArchiver *)archiver didEncodeObject:(id)object
```

Parameters

archiver

The archiver that sent the message.

object

The object that has been encoded. *object* may be `nil`.

Discussion

The delegate might restore some state it had modified previously, or use this opportunity to keep track of the objects that are encoded.

This method is not called for conditional objects until they are actually encoded (if ever).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyedArchiver.h`

archiver:willEncodeObject:

Informs the delegate that *object* is about to be encoded.

```
- (id)archiver:(NSKeyedArchiver *)archiver willEncodeObject:(id)object
```

Parameters

archiver

The archiver that sent the message.

object

The object that is about to be encoded. This value is never `nil`.

Return Value

Either *object* or a different object to be encoded in its stead. The delegate can also modify the coder state. If the delegate returns `nil`, `nil` is encoded.

Discussion

This method is called after the original object may have replaced itself with `replacementObjectForKeyedArchiver:` (page 814).

This method is called whether or not the object is being encoded conditionally.

This method is not called for an object once a replacement mapping has been set up for that object (either explicitly, or because the object has previously been encoded). This method is also not called when `nil` is about to be encoded.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

archiver:willReplaceObject:withObject:

Informs the delegate that one given object is being substituted for another given object.

```
- (void)archiver:(NSKeyedArchiver *)archiver willReplaceObject:(id)object
    withObject:(id)newObject
```

Parameters

archiver
The archiver that sent the message.

object
The object being replaced in the archive.

newObject
The object replacing *object* in the archive.

Discussion
This method is called even when the delegate itself is doing, or has done, the substitution. The delegate may use this method if it is keeping track of the encoded or decoded objects.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

archiverDidFinish:

Notifies the delegate that encoding has finished.

```
- (void)archiverDidFinish:(NSKeyedArchiver *)archiver
```

Parameters

archiver
The archiver that sent the message.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

archiverWillFinish:

Notifies the delegate that encoding is about to finish.

```
- (void)archiverWillFinish:(NSKeyedArchiver *)archiver
```

Parameters

archiver

The archiver that sent the message.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSKeyedArchiver.h

Constants

Keyed Archiving Exception Names

Names of exceptions that are raised by `NSKeyedArchiver` if there is a problem creating an archive.

```
extern NSString *NSInvalidArchiveOperationException;
```

Constants

`NSInvalidArchiveOperationException`

The name of the exception raised by `NSKeyedArchiver` if there is a problem creating an archive.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyedArchiver.h`

Availability

Available in Mac OS X v10.2 and later.

Declared In

NSKeyedArchiver.h

NSKeyedUnarchiver Class Reference

Inherits from:	NSCoder : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSKeyedArchiver.h
Companion guide:	Archives and Serializations Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSKeyedUnarchiver`, a concrete subclass of `NSCoder`, defines methods for decoding a set of named objects (and scalar values) from a keyed archive. Such archives are produced by instances of the `NSKeyedArchiver` class.

A keyed archive is encoded as a hierarchy of objects. Each object in the hierarchy serves as a namespace into which other objects are encoded. The objects available for decoding are restricted to those that were encoded within the immediate scope of a particular object. Objects encoded elsewhere in the hierarchy, whether higher than, lower than, or parallel to this particular object, are not accessible. In this way, the keys used by a particular object to encode its instance variables need to be unique only within the scope of that object.

If you invoke one of the `decode...` methods of this class using a key that does not exist in the archive, a non-positive value is returned. This value varies by decoded type. For example, if a key does not exist in an archive, `decodeBoolForKey:` (page 522) returns `NO`, `decodeIntForKey:` (page 525) returns `0`, and `decodeObjectForKey:` (page 526) returns nil.

`NSKeyedUnarchiver` supports limited type coercion. A value encoded as any type of integer, whether a standard `int` or an explicit 32-bit or 64-bit integer, can be decoded using any of the integer decode methods. Likewise, a value encoded as a `float` or `double` can be decoded as either a `float` or a `double` value. If an encoded value is too large to fit within the coerced type, the decoding method raises an `NSRangeException`. Further, when trying to coerce a value to an incompatible type, for example decoding an `int` as a `float`, the decoding method raises an `NSInvalidUnarchiveOperationException`.

Tasks

Initializing a Keyed Unarchiver

`- initForReadingWithData:` (page 527)
Initializes the receiver for decoding an archive previously encoded by `NSKeyedArchiver`.

Unarchiving Data

`+ unarchiveObjectWithData:` (page 521)
Decodes and returns the object graph previously encoded by `NSKeyedArchiver` and stored in a given `NSData` object.

`+ unarchiveObjectWithFile:` (page 521)
Decodes and returns the object graph previously encoded by `NSKeyedArchiver` written to the file at a given path.

Decoding Data

`- containsValueForKey:` (page 522)
Returns a Boolean value that indicates whether the archive contains a value for a given key within the current decoding scope.

`- decodeBoolForKey:` (page 522)
Decodes a Boolean value associated with a given key.

`- decodeBytesForKey:returnedLength:` (page 523)
Decodes a stream of bytes associated with a given key.

`- decodeDoubleForKey:` (page 523)
Decodes a double-precision floating-point value associated with a given key.

`- decodeFloatForKey:` (page 524)
Decodes a single-precision floating-point value associated with a given key.

`- decodeIntForKey:` (page 525)
Decodes an integer value associated with a given key.

- `decodeInt32ForKey:` (page 524)
  Decodes a 32-bit integer value associated with a given key.
- `decodeInt64ForKey:` (page 525)
  Decodes a 64-bit integer value associated with a given key.
- `decodeObjectForKey:` (page 526)
  Decodes and returns an object associated with a given key.
- `finishDecoding` (page 527)
  Tells the receiver that you are finished decoding objects.

Managing the Delegate

- `delegate` (page 526)
  Returns the receiver's delegate.
- `setDelegate:` (page 528)
  Sets the receiver's delegate.

Managing Class Names

+ `setClass:forClassName:` (page 520)
  Adds a class translation mapping to `NSKeyedUnarchiver` whereby objects encoded with a given class name are decoded as instances of a given class instead.
+ `classForClassName:` (page 520)
  Returns the class from which `NSKeyedUnarchiver` instantiates an encoded object with a given class name.
- `setClass:forClassName:` (page 527)
  Adds a class translation mapping to the receiver whereby objects encoded with a given class name are decoded as instances of a given class instead.
- `classForClassName:` (page 522)
  Returns the class from which the receiver instantiates an encoded object with a given class name.

Decoding Objects

- `unarchiver:cannotDecodeObjectOfClassName:originalClasses:` (page 528) *delegate method*
  Informs the delegate that the class with a given name is not available during decoding.
- `unarchiver:didDecodeObject:` (page 529) *delegate method*
  Informs the delegate that a given object has been decoded.
- `unarchiver:willReplaceObject:withObject:` (page 529) *delegate method*
  Informs the delegate that one object is being substituted for another.

Finishing Decoding

- `unarchiverDidFinish:` (page 530) *delegate method*
  Notifies the delegate that decoding has finished.

- unarchiverWillFinish: (page 530) *delegate method*
  Notifies the delegate that decoding is about to finish.

Class Methods

classForClassName:

Returns the class from which NSKeyedUnarchiver instantiates an encoded object with a given class name.

```
+ (Class)classForClassName:(NSString *)codedName
```

Parameters

codedName
The ostensible name of a class in an archive.

Return Value

The class from which NSKeyedUnarchiver instantiates an object encoded with the class name *codedName*. Returns nil if NSKeyedUnarchiver does not have a translation mapping for *codedName*.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ setClass:forClassName: (page 520)
- classForClassName: (page 522)

Declared In

NSKeyedArchiver.h

setClass:forClassName:

Adds a class translation mapping to NSKeyedUnarchiver whereby objects encoded with a given class name are decoded as instances of a given class instead.

```
+ (void)setClass:(Class)cls forClassName:(NSString *)codedName
```

Parameters

cls
The class with which to replace instances of the class named *codedName*.

codedName
The ostensible name of a class in an archive.

Discussion

When decoding, the class's translation mapping is used only if no translation is found first in an instance's separate translation map.

Availability

Available in iPhone OS 2.0 and later.

See Also
+ classForClassName: (page 520)
- setClass:forClassName: (page 527)

Declared In
NSKeyedArchiver.h

unarchiveObjectWithData:

Decodes and returns the object graph previously encoded by NSKeyedArchiver and stored in a given NSData object.

```
+ (id)unarchiveObjectWithData:(NSData *)data
```

Parameters

data

An object graph previously encoded by NSKeyedArchiver.

Return Value
The object graph previously encoded by NSKeyedArchiver and stored in *data*.

Discussion
This method raises an NSInvalidArchiveOperationException (page 515) if *data* is not a valid archive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

unarchiveObjectWithFile:

Decodes and returns the object graph previously encoded by NSKeyedArchiver written to the file at a given path.

```
+ (id)unarchiveObjectWithFile:(NSString *)path
```

Parameters

path

A path to a file that contains an object graph previously encoded by NSKeyedArchiver.

Return Value
The object graph previously encoded by NSKeyedArchiver written to the file *path*. Returns nil if there is no file at *path*.

Discussion
This method raises an NSInvalidArgumentException if the file at *path* does not contain a valid archive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

Instance Methods

classForClassName:

Returns the class from which the receiver instantiates an encoded object with a given class name.

```
- (Class)classForClassName:(NSString *)codedName
```

Parameters

codedName

The name of a class.

Return Value

The class from which the receiver instantiates an encoded object with the class name *codedName*. Returns `nil` if the receiver does not have a translation mapping for *codedName*.

Discussion

The class's separate translation map is not searched.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setClass:forClassName:` (page 527)

`+ classForClassName:` (page 520)

Declared In

`NSKeyedArchiver.h`

containsValueForKey:

Returns a Boolean value that indicates whether the archive contains a value for a given key within the current decoding scope.

```
- (BOOL)containsValueForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

`YES` if the archive contains a value for *key* within the current decoding scope, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyedArchiver.h`

decodeBoolForKey:

Decodes a Boolean value associated with a given key.

```
- (BOOL)decodeBoolForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

The Boolean value associated with the key *key*. Returns `NO` if *key* does not exist.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeBool:forKey:` (page 506) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeBytesForKey:returnedLength:

Decodes a stream of bytes associated with a given key.

```
- (const uint8_t *)decodeBytesForKey:(NSString *)key returnedLength:(NSUInteger
    *)lengthp
```

Parameters

key

A key in the archive within the current decoding scope.

lengthp

Upon return, contains the number of bytes returned.

Return Value

The stream of bytes associated with the key *key*. Returns `NULL` if *key* does not exist.

Discussion

The returned value is a pointer to a temporary buffer owned by the receiver. The buffer goes away with the unarchiver, not the containing autorelease pool. You must copy the bytes into your own buffer if you need the data to persist beyond the life of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- encodeBytes:length:forKey:` (page 507) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeDoubleForKey:

Decodes a double-precision floating-point value associated with a given key.

```
- (double)decodeDoubleForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

The double-precision floating-point value associated with the key *key*. Returns `0.0` if *key* does not exist.

Discussion

If the archived value was encoded as single-precision, the type is coerced.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `encodeDouble:forKey:` (page 508) (`NSKeyedArchiver`)
- `encodeFloat:forKey:` (page 508) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeFloatForKey:

Decodes a single-precision floating-point value associated with a given key.

```
- (float)decodeFloatForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

The single-precision floating-point value associated with the key *key*. Returns `0.0` if *key* does not exist.

Discussion

If the archived value was encoded as double precision, the type is coerced, loosing precision. If the archived value is too large for single precision, the method raises an `NSRangeException`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `encodeFloat:forKey:` (page 508) (`NSKeyedArchiver`)
- `encodeDouble:forKey:` (page 508) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeInt32ForKey:

Decodes a 32-bit integer value associated with a given key.

```
- (int32_t)decodeInt32ForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

The 32-bit integer value associated with the key *key*. Returns 0 if *key* does not exist.

Discussion

If the archived value was encoded with a different size but is still an integer, the type is coerced. If the archived value is too large to fit into a 32-bit integer, the method raises an `NSRangeException`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `encodeInt32:forKey:` (page 509) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeInt64ForKey:

Decodes a 64-bit integer value associated with a given key.

```
- (int64_t)decodeInt64ForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value

The 64-bit integer value associated with the key *key*. Returns 0 if *key* does not exist.

Discussion

If the archived value was encoded with a different size but is still an integer, the type is coerced.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `encodeInt64:forKey:` (page 509) (`NSKeyedArchiver`)

Declared In

`NSKeyedArchiver.h`

decodeIntForKey:

Decodes an integer value associated with a given key.

```
- (int)decodeIntForKey:(NSString *)key
```

Parameters

key

A key in the archive within the current decoding scope.

Return Value
The integer value associated with the key `key`. Returns `0` if *`key`* does not exist.

Discussion
If the archived value was encoded with a different size but is still an integer, the type is coerced. If the archived value is too large to fit into the default size for an integer, the method raises an `NSRangeException`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- encodeInt:forKey:` (page 509) (`NSKeyedArchiver`)

Declared In
`NSKeyedArchiver.h`

decodeObjectForKey:

Decodes and returns an object associated with a given key.

```
- (id)decodeObjectForKey:(NSString *)key
```

Parameters

`key`
A key in the archive within the current decoding scope.

Return Value
The object associated with the key *`key`*. Returns `nil` if *`key`* does not exist.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- encodeObject:forKey:` (page 510) (`NSKeyedArchiver`)

Declared In
`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setDelegate:` (page 528)

Declared In
NSKeyedArchiver.h

finishDecoding

Tells the receiver that you are finished decoding objects.

```
- (void)finishDecoding
```

Discussion
Invoking this method allows the receiver to notify its delegate and to perform any final operations on the archive. Once this method is invoked, the receiver cannot decode any further values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

initForReadingWithData:

Initializes the receiver for decoding an archive previously encoded by `NSKeyedArchiver`.

```
- (id)initForReadingWithData:(NSData *)data
```

Parameters
data
An archive previously encoded by `NSKeyedArchiver`.

Return Value
An `NSKeyedUnarchiver` object initialized for for decoding *data*.

Discussion
When you finish decoding data, you should invoke `finishDecoding` (page 527).

This method raises an NSInvalidArchiveOperationException (page 515) if *data* is not a valid archive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSKeyedArchiver.h

setClass:forClassName:

Adds a class translation mapping to the receiver whereby objects encoded with a given class name are decoded as instances of a given class instead.

```
- (void)setClass:(Class)cls forClassName:(NSString *)codedName
```

Parameters
cls
The class with which to replace instances of the class named *codedName*.

codedName
The ostensible name of a class in an archive.

Discussion
When decoding, the receiver's translation map overrides any translation that may also be present in the class's map (see setClass:forClassName: (page 520)).

Availability
Available in iPhone OS 2.0 and later.

See Also
- classForClassName: (page 522)
+ setClass:forClassName: (page 520)

Declared In
NSKeyedArchiver.h

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters
delegate
The delegate for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- delegate (page 526)

Declared In
NSKeyedArchiver.h

Delegate Methods

unarchiver:cannotDecodeObjectOfClassName:originalClasses:

Informs the delegate that the class with a given name is not available during decoding.

```
- (Class)unarchiver:(NSKeyedUnarchiver *)unarchiver
    cannotDecodeObjectOfClassName:(NSString *)name originalClasses:(NSArray
    *)classNames
```

Parameters
unarchiver
An unarchiver for which the receiver is the delegate.

name
The name of the class of an object `unarchiver` is trying to decode.

classNames
An array describing the class hierarchy of the encoded object, where the first element is the class name string of the encoded object, the second element is the class name of its immediate superclass, and so on.

Return Value
The class unarchiver should use in place of the class named *name*.

Discussion
The delegate may, for example, load some code to introduce the class to the runtime and return the class, or substitute a different class object. If the delegate returns `nil`, unarchiving aborts and the method raises an `NSInvalidUnarchiveOperationException`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

unarchiver:didDecodeObject:

Informs the delegate that a given object has been decoded.

```
- (id)unarchiver:(NSKeyedUnarchiver *)unarchiver didDecodeObject:(id)object
```

Parameters
unarchiver
An unarchiver for which the receiver is the delegate.

object
The object that has been decoded. *object* may be `nil`.

Return Value
The object to use in place of *object*. The delegate can either return *object* or return a different object to replace the decoded one. If the delegate returns `nil`, `nil` is the result of decoding *object*.

Discussion
This method is called after *object* has been sent `initWithCoder:` (page 1246) and `awakeAfterUsingCoder:` (page 797).

The delegate may use this method to keep track of the decoded objects.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

unarchiver:willReplaceObject:withObject:

Informs the delegate that one object is being substituted for another.

```
- (void)unarchiver:(NSKeyedUnarchiver *)unarchiver willReplaceObject:(id)object
    withObject:(id)newObject
```

Parameters

unarchiver
An unarchiver for which the receiver is the delegate.

object
An object in the archive.

newObject
The object with which *unarchiver* will replace *object*.

Discussion
This method is called even when the delegate itself is doing, or has done, the substitution with `unarchiver:didDecodeObject:` (page 529).

The delegate may use this method if it is keeping track of the encoded or decoded objects.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

unarchiverDidFinish:

Notifies the delegate that decoding has finished.

```
- (void)unarchiverDidFinish:(NSKeyedUnarchiver *)unarchiver
```

Parameters

unarchiver
An unarchiver for which the receiver is the delegate.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

unarchiverWillFinish:

Notifies the delegate that decoding is about to finish.

```
- (void)unarchiverWillFinish:(NSKeyedUnarchiver *)unarchiver
```

Parameters

unarchiver
An unarchiver for which the receiver is the delegate.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

Constants

Keyed Unarchiving Exception Names

Names of exceptions that are raised by `NSKeyedUnarchiver` if there is a problem extracting an archive.

```
extern NSString *NSInvalidUnarchiveOperationException;
```

Constants

`NSInvalidUnarchiveOperationException`

The name of the exception raised by `NSKeyedArchiver` if there is a problem extracting an archive.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyedArchiver.h`

Declared In

`NSKeyedUnarchiver.h`

NSLocale Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLocale.h
Companion guides:	Locales Data Formatting Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

Locales encapsulate information about linguistic, cultural, and technological conventions and standards. Examples of information encapsulated by a locale include the symbol used for the decimal separator in numbers and the way dates are formatted.

Locales are typically used to provide, format, and interpret information about and according to the user's customs and preferences. They are frequently used in conjunction with formatters (see *Data Formatting Programming Guide for Cocoa*). Although you can use many locales, you usually use the one associated with the current user.

`NSLocale` is "toll-free bridged" with its Core Foundation counterpart, `CFLocale`. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSLocale *` parameter, you can pass a `CFLocaleRef`, and in a function where you see a `CFLocaleRef` parameter, you can pass an `NSLocale` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Getting and Initializing Locales

`- initWithLocaleIdentifier:` (page 542)
Initializes the receiver using a given locale identifier.

`+ systemLocale` (page 540)
Returns the "root", canonical locale, that contains fixed "backstop" settings that provide values for otherwise undefined keys.

`+ currentLocale` (page 537)
Returns the logical locale for the current user.

`+ autoupdatingCurrentLocale` (page 535)
Returns the current logical locale for the current user.

Getting Information About a Locale

`- displayNameForKey:value:` (page 541)
Returns the display name for the given value.

`- localeIdentifier` (page 542)
Returns the identifier for the receiver.

`- objectForKey:` (page 543)
Returns the object corresponding to the specified key.

Getting System Locale Information

`+ availableLocaleIdentifiers` (page 536)
Returns an array of `NSString` objects, each of which identifies a locale available on the system.

`+ ISOCountryCodes` (page 538)
Returns an array of `NSString` objects that represents all known legal country codes.

`+ ISOCurrencyCodes` (page 538)
Returns an array of `NSString` objects that represents all known legal ISO currency codes.

`+ ISOLanguageCodes` (page 539)
Returns an array of `NSString` objects that represents all known legal ISO language codes.

`+ commonISOCurrencyCodes` (page 536)
Returns an array of common ISO currency codes

Converting Between Identifiers

`+ canonicalLocaleIdentifierFromString:` (page 536)
Returns the canonical identifier for a given locale identification string.

`+ componentsFromLocaleIdentifier:` (page 537)
Returns a dictionary that is the result of parsing a locale ID.

`+ localeIdentifierFromComponents:` (page 539)
Returns a locale identifier from the components specified in a given dictionary.

Getting Preferred Languages

`+ preferredLanguages` (page 540)
Returns the user's language preference order as an array of strings.

Class Methods

autoupdatingCurrentLocale

Returns the current logical locale for the current user.

```
+ (id)autoupdatingCurrentLocale
```

Return Value

The current logical locale for the current user. The locale is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences.

The object always reflects the current state of the current user's locale settings.

Discussion

Settings you get from this locale do change as the user's settings change (contrast with `currentLocale` (page 537)).

Note that if you cache values based on the locale or related information, those caches will of course not be automatically updated by the updating of the locale object. You can recompute caches upon receipt of the notification (`NSCurrentLocaleDidChangeNotification`) that gets sent out for locale changes (see *Notification Programming Topics for Cocoa* to learn how to register for and receive notifications).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ systemLocale` (page 540)
`+ currentLocale` (page 537)

Declared In

`NSLocale.h`

availableLocaleIdentifiers

Returns an array of `NSString` objects, each of which identifies a locale available on the system.

```
+ (NSArray *)availableLocaleIdentifiers
```

Return Value

An array of `NSString` objects, each of which identifies a locale available on the system.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ `ISOLanguageCodes` (page 539)
+ `ISOCountryCodes` (page 538)
+ `ISOCurrencyCodes` (page 538)
+ `commonISOCurrencyCodes` (page 536)

Declared In

`NSLocale.h`

canonicalLocaleIdentifierFromString:

Returns the canonical identifier for a given locale identification string.

```
+ (NSString *)canonicalLocaleIdentifierFromString:(NSString *)string
```

Parameters

string

A locale identification string.

Return Value

The canonical identifier for an the locale identified by *string*.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ `componentsFromLocaleIdentifier:` (page 537)
+ `localeIdentifierFromComponents:` (page 539)

Declared In

`NSLocale.h`

commonISOCurrencyCodes

Returns an array of common ISO currency codes

```
+ (NSArray *)commonISOCurrencyCodes
```

Return Value

An array of `NSString` objects that represents common ISO currency codes.

Discussion
Common codes may include, for example, AED, AUD, BZD, DKK, EUR, GBP, JPY, KES, MXN, OMR, STD, USD, XCD, and ZWD.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ availableLocaleIdentifiers (page 536)
+ ISOCountryCodes (page 538)
+ ISOCurrencyCodes (page 538)

Declared In
NSLocale.h

componentsFromLocaleIdentifier:
Returns a dictionary that is the result of parsing a locale ID.

```
+ (NSDictionary *)componentsFromLocaleIdentifier:(NSString *)string
```

Parameters
string
A locale ID, consisting of language, script, country, variant, and keyword/value pairs, for example, "en_US@calendar=japanese".

Return Value
A dictionary that is the result of parsing *string* as a locale ID. The keys are the constant NSString constants corresponding to the locale ID components, and the values correspond to constants where available. For the complete set of dictionary keys, see "Constants" (page 543).

Discussion
For example: the locale ID "en_US@calendar=japanese" yields a dictionary with three entries: NSLocaleLanguageCode=en, NSLocaleCountryCode=US, and NSLocaleCalendar=NSJapaneseCalendar.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ localeIdentifierFromComponents: (page 539)
+ canonicalLocaleIdentifierFromString: (page 536)

Declared In
NSLocale.h

currentLocale
Returns the logical locale for the current user.

```
+ (id)currentLocale
```

Return Value
The logical locale for the current user. The locale is formed from the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences.

This method may return a retained cached object.

Discussion
Settings you get from this locale do not change as System Preferences are changed so that your operations are consistent. Typically you perform some operations on the returned object and then allow it to be disposed of. Moreover, since the returned object may be cached, you do not need to hold on to it indefinitely. Contrast with `autoupdatingCurrentLocale` (page 535).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ systemLocale` (page 540)
`+ autoupdatingCurrentLocale` (page 535)

Declared In
`NSLocale.h`

ISOCountryCodes
Returns an array of `NSString` objects that represents all known legal country codes.

```
+ (NSArray *)ISOCountryCodes
```

Return Value
An array of `NSString` objects that represents all known legal country codes.

Discussion
Note that many of country codes do not have any supporting locale data in Mac OS X.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ availableLocaleIdentifiers` (page 536)
`+ ISOLanguageCodes` (page 539)
`+ ISOCurrencyCodes` (page 538)
`+ commonISOCurrencyCodes` (page 536)

Declared In
`NSLocale.h`

ISOCurrencyCodes
Returns an array of `NSString` objects that represents all known legal ISO currency codes.

```
+ (NSArray *)ISOCurrencyCodes
```

Return Value
An array of `NSString` objects that represents all known legal ISO currency codes.

Discussion
Note that some of the currency codes may not have any supporting locale data in Mac OS X.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ availableLocaleIdentifiers` (page 536)
`+ ISOCountryCodes` (page 538)
`+ ISOLanguageCodes` (page 539)
`+ commonISOCurrencyCodes` (page 536)

Declared In
`NSLocale.h`

ISOLanguageCodes
Returns an array of `NSString` objects that represents all known legal ISO language codes.

```
+ (NSArray *)ISOLanguageCodes
```

Return Value
An array of `NSString` objects that represents all known legal ISO language codes.

Discussion
Note that many of the language codes will not have any supporting locale data in Mac OS X.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ availableLocaleIdentifiers` (page 536)
`+ ISOCountryCodes` (page 538)
`+ ISOCurrencyCodes` (page 538)
`+ commonISOCurrencyCodes` (page 536)

Declared In
`NSLocale.h`

localeIdentifierFromComponents:
Returns a locale identifier from the components specified in a given dictionary.

```
+ (NSString *)localeIdentifierFromComponents:(NSDictionary *)dict
```

Parameters
dict

A dictionary containing components that specify a locale. For valid dictionary keys, see "Constants" (page 543).

Return Value
A locale identifier created from the components specified in *dict*.

Discussion
This reverses the actions of `componentsFromLocaleIdentifier:` (page 537), so for example the dictionary `{NSLocaleLanguageCode="en", NSLocaleCountryCode="US", NSLocaleCalendar=NSJapaneseCalendar}` becomes `"en_US@calendar=japanese"`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ componentsFromLocaleIdentifier:` (page 537)
`+ canonicalLocaleIdentifierFromString:` (page 536)
`+ ISOLanguageCodes` (page 539)

Declared In
`NSLocale.h`

preferredLanguages

Returns the user's language preference order as an array of strings.

```
+ (NSArray *)preferredLanguages
```

Return Value
The user's language preference order as an array of `NSString` objects, each of which is a canonicalized IETF BCP 47 language identifier.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLocale.h`

systemLocale

Returns the “root”, canonical locale, that contains fixed “backstop” settings that provide values for otherwise undefined keys.

```
+ (id)systemLocale
```

Return Value
The “root”, canonical locale, that contains fixed “backstop” settings that provide values for otherwise undefined keys.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ autoupdatingCurrentLocale` (page 535)
`+ autoupdatingCurrentLocale` (page 535)

Declared In
NSLocale.h

Instance Methods

displayNameForKey:value:

Returns the display name for the given value.

```
- (NSString *)displayNameForKey:(id)key value:(id)value
```

Parameters

key

Specifies which of the locale property keys *value* is (see "Constants" (page 543)),

value

A value for *key*.

Return Value
The display name for *value*.

Discussion
Not all locale property keys have values with display name values.

You can use the NSLocaleIdentifier key to get the name of a locale in the language of another locale, as illustrated in the following examples. The first uses the fr_FR locale.

```
NSLocale *frLocale = [[[NSLocale alloc] initWithLocaleIdentifier:@"fr_FR"]
autorelease];
NSString *displayNameString = [frLocale displayNameForKey:NSLocaleIdentifier
value:@"fr_FR"];
NSLog(@"displayNameString fr_FR: %@", displayNameString);
displayNameString = [frLocale displayNameForKey:NSLocaleIdentifier
value:@"en_US"];
NSLog(@"displayNameString en_US: %@", displayNameString);
```

returns

```
displayNameString fr_FR: français (France)
displayNameString en_US: anglais (États-Unis)
```

The following example uses the en_GB locale.

```
NSLocale *gbLocale = [[[NSLocale alloc] initWithLocaleIdentifier:@"en_GB"]
autorelease];
displayNameString = [gbLocale displayNameForKey:NSLocaleIdentifier
value:@"fr_FR"];
NSLog(@"displayNameString fr_FR: %@", displayNameString);
displayNameString = [gbLocale displayNameForKey:NSLocaleIdentifier
value:@"en_US"];
NSLog(@"displayNameString en_US: %@", displayNameString);
```

returns

```
displayNameString fr_FR: French (France)
```

```
displayNameString en_US: English (United States)
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `localeIdentifier` (page 542)

Declared In
`NSLocale.h`

initWithLocaleIdentifier:

Initializes the receiver using a given locale identifier.

```
- (id)initWithLocaleIdentifier:(NSString *)string
```

Parameters
string
The identifier for the new locale.

Return Value
The initialized locale.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLocale.h`

localeIdentifier

Returns the identifier for the receiver.

```
- (NSString *)localeIdentifier
```

Return Value
The identifier for the receiver. This may not be the same string that the locale was created with, since `NSLocale` may canonicalize it.

Discussion
Equivalent to sending `objectForKey:` with key `NSLocaleIdentifier`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `displayNameForKey:value:` (page 541)

Declared In
`NSLocale.h`

objectForKey:

Returns the object corresponding to the specified key.

```
- (id)objectForKey:(id)key
```

Parameters

key

The key for which to return the corresponding value. For valid values of *key*, see "Constants" (page 543).

Return Value

The object corresponding to *key*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- displayNameForKey:value: (page 541)

Declared In

NSLocale.h

Constants

NSLocale Component Keys

The following constants specify keys used to retrieve components of a locale with objectForKey: (page 543).

```
extern NSString * const NSLocaleIdentifier;
extern NSString * const NSLocaleLanguageCode;
extern NSString * const NSLocaleCountryCode;
extern NSString * const NSLocaleScriptCode;
extern NSString * const NSLocaleVariantCode;
extern NSString * const NSLocaleExemplarCharacterSet;
extern NSString * const NSLocaleCalendar;
extern NSString * const NSLocaleCollationIdentifier;
extern NSString * const NSLocaleUsesMetricSystem;
extern NSString * const NSLocaleMeasurementSystem;
extern NSString * const NSLocaleDecimalSeparator;
extern NSString * const NSLocaleGroupingSeparator;
extern NSString * const NSLocaleCurrencySymbol;
extern NSString * const NSLocaleCurrencyCode;
```

Constants

NSLocaleIdentifier

The key for the locale identifier.

The corresponding value is an NSString object. An example value might be "es_ES_PREEURO".

Available in iPhone OS 2.0 and later.

Declared in NSLocale.h

`NSLocaleLanguageCode`
The key for the locale language code.
The corresponding value is an `NSString` object. An example value might be `"es"`.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleCountryCode`
The key for the locale country code.
The corresponding value is an `NSString` object. An example value might be `"ES"`.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleScriptCode`
The key for the locale script code.
The corresponding value is an `NSString` object.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleVariantCode`
The key for the locale variant code.
The corresponding value is an `NSString` object. An example value might be `"PREEURO"`.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleExemplarCharacterSet`
The key for the exemplar character set for the locale.
The corresponding value is an `NSCharacterSet` object.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleCalendar`
The key for the calendar associated with the locale.
The corresponding value is an `NSCalendar` object.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleCollationIdentifier`
The key for the collation associated with the locale.
The corresponding value is an `NSString` object.
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleUsesMetricSystem`
The key for the flag that indicates whether the locale uses the metric system.
The corresponding value is a Boolean `NSNumber` object. If the value is `NO`, you can typically assume American measurement units (for example, the statute mile).
Available in iPhone OS 2.0 and later.
Declared in `NSLocale.h`

`NSLocaleMeasurementSystem`
The key for the measurement system associated with the locale.

The corresponding value is an `NSString` object containing a description of the measurement system used by the locale, for example “Metric” or “U.S.”.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSLocaleDecimalSeparator`
The key for the decimal separator associated with the locale.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSLocaleGroupingSeparator`
The key for the numeric grouping separator associated with the locale.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSLocaleCurrencySymbol`
The key for the currency symbol associated with the locale.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSLocaleCurrencyCode`
The key for the currency code associated with the locale.

The corresponding value is an `NSString` object.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

Declared In
`NSLocale.h`

NSLocale Calendar Keys

These constants identify `NSCalendar` instances.

```
extern NSString * const NSGregorianCalendar;
extern NSString * const NSBuddhistCalendar;
extern NSString * const NSChineseCalendar;
extern NSString * const NSHebrewCalendar;
extern NSString * const NSIslamicCalendar;
extern NSString * const NSIslamicCivilCalendar;
extern NSString * const NSJapaneseCalendar;
```

Constants
`NSGregorianCalendar`
Identifier for the Gregorian calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSBuddhistCalendar`
Identifier for the Buddhist calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSChineseCalendar`
Identifier for the Chinese calendar (unsupported).

Note that the Chinese calendar is not supported in Mac OS X v10.4-10.5. Although you can create a calendar using this constant, the object will not function correctly.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSHebrewCalendar`
Identifier for the Hebrew calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSIslamicCalendar`
Identifier for the Islamic calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSIslamicCivilCalendar`
Identifier for the Islamic civil calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

`NSJapaneseCalendar`
Identifier for the Japanese calendar.

Available in iPhone OS 2.0 and later.

Declared in `NSLocale.h`

Discussion
You use these identifiers to initialize a new `NSCalendar` object, using `initWithCalendarIdentifier:` (page 121). You get one of these identifiers as the return value from `calendarIdentifier` (page 117).

Declared In
`NSLocale.h`

Notifications

NSCurrentLocaleDidChangeNotification

Notification that indicates that the user's locale changed.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLocale.h`

NSLock Class Reference

Inherits from:	NSObject
Conforms to:	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLock.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSLock` object is used to coordinate the operation of multiple threads of execution within the same application. An `NSLock` object can be used to mediate access to an application's global data or to protect a critical section of code, allowing it to run atomically.

Warning: The `NSLock` class uses POSIX threads to implement its locking behavior. When sending an unlock message to an `NSLock` object, you must be sure that message is sent from the same thread that sent the initial lock message. Unlocking a lock from a different thread can result in undefined behavior.

You should not use this class to implement a recursive lock. Calling the `lock` method twice on the same thread will lock up your thread permanently. Use the `NSRecursiveLock` class to implement recursive locks instead.

Unlocking a lock that is not locked is considered a programmer error and should be fixed in your code. The `NSLock` class reports such errors by printing an error message to the console when they occur.

Adopted Protocols

NSLocking
- `lock` (page 1298)
- `unlock` (page 1298)

Tasks

Acquiring a Lock

- `lockBeforeDate:` (page 548)
  Attempts to acquire a lock before a given time and returns a Boolean value indicating whether the attempt was successful.
- `tryLock` (page 550)
  Attempts to acquire a lock and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

- `setName:` (page 549)
  Assigns a name to the receiver.
- `name` (page 549)
  Returns the name associated with the receiver.

Instance Methods

lockBeforeDate:

Attempts to acquire a lock before a given time and returns a Boolean value indicating whether the attempt was successful.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit

The time limit for attempting to acquire a lock.

Return Value

YES if the lock is acquired before *limit*, otherwise NO.

Discussion

The thread is blocked until the receiver acquires the lock or *limit* is reached.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSLock.h

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value

The name of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setName: (page 549)

Declared In

NSLock.h

setName:

Assigns a name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

newName

The new name for the receiver. This method makes a copy of the specified string.

Discussion

You can use a name string to identify a lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- name (page 549)

Declared In
NSLock.h

tryLock

Attempts to acquire a lock and immediately returns a Boolean value that indicates whether the attempt was successful.

```
- (BOOL)tryLock
```

Return Value
YES if the lock was acquired, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSLock.h

NSMachPort Class Reference

Inherits from:	NSPort : NSObject
Conforms to:	NSCoding (NSPort) NSCopying (NSPort) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSPort.h
Companion guide:	Distributed Objects

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSMachPort` is a subclass of `NSPort` that can be used as an endpoint for distributed object connections (or raw messaging). `NSMachPort` is an object wrapper for a Mach port, the fundamental communication port in Mac OS X. `NSMachPort` allows for local (on the same machine) communication only. A companion class, `NSSocketPort`, allows for both local and remote distributed object communication, but may be more expensive than `NSMachPort` for the local case.

To use `NSMachPort` effectively, you should be familiar with Mach ports, port access rights, and Mach messages. See the Mach OS documentation for more information.

Note: `NSMachPort` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder`. `NSPort` and its subclasses do not support archiving.

Tasks

Creating and Initializing

`+ portWithMachPort:` (page 552)
Creates and returns a port object configured with the given Mach port.

`+ portWithMachPort:options:` (page 553)
Creates and returns a port object configured with the specified options and the given Mach port.

`- initWithMachPort:` (page 553)
Initializes a newly allocated `NSMachPort` object with a given Mach port.

`- initWithMachPort:options:` (page 554)
Initializes a newly allocated `NSMachPort` object with a given Mach port and the specified options.

Getting the Mach Port

`- machPort` (page 554)
Returns as an `int` the Mach port used by the receiver.

Scheduling the Port on a Run Loop

`- removeFromRunLoop:forMode:` (page 555)
Removes the receiver from the run loop mode *`mode`* of *`runLoop`*.

`- scheduleInRunLoop:forMode:` (page 555)
Schedules the receiver into the run loop mode *`mode`* of *`runLoop`*.

Handling Mach Messages

`- handleMachMessage:` (page 556) *delegate method*
Process an incoming Mach message.

Class Methods

portWithMachPort:

Creates and returns a port object configured with the given Mach port.

```
+ (NSPort *)portWithMachPort:(uint32_t)machPort
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

Return Value

An `NSMachPort` object that uses *machPort* to send or receive messages.

Discussion

Creates the port object if necessary. Depending on the access rights associated with *machPort*, the new port object may be usable only for sending messages.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

portWithMachPort:options:

Creates and returns a port object configured with the specified options and the given Mach port.

```
+ (NSPort *)portWithMachPort:(uint32_t)machPort options:(NSUInteger)options
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

options

Specifies options for what to do with the underlying port rights when the `NSMachPort` object is invalidated or destroyed. For a list of constants, see "Mach Port Rights" (page 556).

Return Value

An `NSMachPort` object that uses *machPort* to send or receive messages.

Discussion

Creates the port object if necessary. Depending on the access rights associated with *machPort*, the new port object may be usable only for sending messages.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

Instance Methods

initWithMachPort:

Initializes a newly allocated `NSMachPort` object with a given Mach port.

```
- (id)initWithMachPort:(uint32_t)machPort
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

Return Value

Returns an initialized `NSMachPort` object that uses *machPort* to send or receive messages. The returned object might be different than the original receiver

Discussion

Depending on the access rights for *machPort*, the new port may be able to only send messages. If a port with *machPort* already exists, this method deallocates the receiver, then retains and returns the existing port.

This method is the designated initializer for the `NSMachPort` class.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

initWithMachPort:options:

Initializes a newly allocated `NSMachPort` object with a given Mach port and the specified options.

```
- (id)initWithMachPort:(uint32_t)machPort options:(NSUInteger)options
```

Parameters

machPort

The Mach port for the new port. This parameter should originally be of type `mach_port_t`.

options

Specifies options for what to do with the underlying port rights when the `NSMachPort` object is invalidated or destroyed. For a list of constants, see "Mach Port Rights" (page 556).

Return Value

Returns an initialized `NSMachPort` object that uses *machPort* to send or receive messages. The returned object might be different than the original receiver

Discussion

Depending on the access rights for *machPort*, the new port may be able to only send messages. If a port with *machPort* already exists, this method deallocates the receiver, then retains and returns the existing port.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

machPort

Returns as an `int` the Mach port used by the receiver.

```
- (uint32_t)machPort
```

Return Value
The Mach port used by the receiver. Cast this value to a `mach_port_t` when using it with Mach system calls.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSPort.h`

removeFromRunLoop:forMode:

Removes the receiver from the run loop mode *mode* of *runLoop*.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters
runLoop
The run loop from which to remove the receiver.

mode
The run loop mode from which to remove the receiver.

Discussion
When the receiver is removed, the run loop stops monitoring the Mach port for incoming messages.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- scheduleInRunLoop:forMode:` (page 555)

Declared In
`NSPort.h`

scheduleInRunLoop:forMode:

Schedules the receiver into the run loop mode *mode* of *runLoop*.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters
runLoop
The run loop to which to add the receiver.

mode
The run loop mode in which to add the receiver.

Discussion
When the receiver is scheduled, the run loop monitors the mach port for incoming messages and, when a message arrives, invokes the delegate method `handleMachMessage:` (page 556).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeFromRunLoop:forMode:` (page 555)

Declared In
`NSPort.h`

Delegate Methods

handleMachMessage:

Process an incoming Mach message.

```
- (void)handleMachMessage:(void *)machMessage
```

Parameters

machMessage

A pointer to a Mach message, cast as a pointer to `void`.

Discussion

The delegate should interpret this data as a pointer to a Mach message beginning with a `msg_header_t` structure and should handle the message appropriately.

The delegate should implement only one of `handleMachMessage:` and `handlePortMessage:` (page 857).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

Constants

Mach Port Rights

Used to remove access rights to a mach port when the `NSMachPort` object is invalidated or destroyed.

```
enum {
    NSMachPortDeallocateNone = 0,
    NSMachPortDeallocateSendRight = (1 << 0),
    NSMachPortDeallocateReceiveRight = (1 << 1)
};
```

Constants

`NSMachPortDeallocateNone`

Do not remove any send or receive rights.

Available in iPhone OS 2.0 and later.

Declared in `NSPort.h`

`NSMachPortDeallocateSendRight`

Deallocate a send right when the `NSMachPort` object is invalidated or destroyed.

Available in iPhone OS 2.0 and later.

Declared in `NSPort.h`

`NSMachPortDeallocateReceiveRight`

Remove a receive right when the `NSMachPort` object is invalidated or destroyed.

Available in iPhone OS 2.0 and later.

Declared in `NSPort.h`

Declared In
`NSPort.h`

NSMessagePort Class Reference

Inherits from:	NSPort : NSObject
Conforms to:	NSCoding (NSPort) NSCopying (NSPort) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSPort.h
Companion guide:	Distributed Objects

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSMessagePort` is a subclass of `NSPort` that can be used as an endpoint for distributed object connections (or raw messaging). `NSMessagePort` allows for local (on the same machine) communication only. A companion class, `NSSocketPort`, allows for both local and remote communication, but may be more expensive than `NSMessagePort` for the local case.

`NSMessagePort` defines no additional methods over those already defined by `NSPort`.

Note: `NSMessagePort` conforms to the `NSCoding` protocol, but only supports coding by an `NSPortCoder` object. `NSPort` and its subclasses do not support archiving.

NSMethodSignature Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSMethodSignature.h
Companion guides:	Distributed Objects The Objective-C 2.0 Programming Language

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSMethodSignature` object records type information for the arguments and return value of a method. It is used to forward messages that the receiving object does not respond to—most notably in the case of distributed objects. An `NSMethodSignature` object is typically created using `NSObject`'s `methodSignatureForSelector:` (page 805) instance method. It is then used to create an `NSInvocation` object, which is passed as the argument to a `forwardInvocation:` (page 801) message to send the invocation on to whatever other object can handle the message. In the default case, `NSObject` invokes `doesNotRecognizeSelector:` (page 799), which raises an exception. For distributed objects, the `NSInvocation` object is encoded using the information in the `NSMethodSignature` object and sent to the real object represented by the receiver of the message.

An `NSMethodSignature` object presents its argument types by index with the `getArgumentTypeAtIndex:` (page 563) method. The hidden arguments for every method, *`self`* and *`_cmd`*, are at indices `0` and `1`, respectively. The arguments normally specified in a message invocation follow these. In addition to the argument types, an `NSMethodSignature` object offers the total number of arguments with `numberOfArguments` (page 564), the total stack frame length occupied by all arguments with `frameLength` (page 562) (this varies with hardware architecture), and the length and type of the return value with `methodReturnLength` (page 564) and `methodReturnType` (page 564). Finally, applications using distributed objects can determine if the method is asynchronous with the `isOneway` (page 563) method.

For more information about the nature of a method, including the hidden arguments, see "How Messaging Works" in "The Language" in *The Objective-C 2.0 Programming Language*.

Tasks

Getting Information on Argument Types

- `getArgumentTypeAtIndex:` (page 563)
  Returns the type encoding for the argument at a given index.
- `numberOfArguments` (page 564)
  Returns the number of arguments recorded in the receiver.
- `frameLength` (page 562)
  Returns the number of bytes that the arguments, taken together, occupy on the stack.

Getting Information on Return Types

- `methodReturnType` (page 564)
  Returns a C string encoding the return type of the method in Objective-C type encoding.
- `methodReturnLength` (page 564)
  Returns the number of bytes required for the return value.

Determining Synchronous Status

- `isOneway` (page 563)
  Returns a Boolean value that indicates whether the receiver is asynchronous when invoked through distributed objects.

Instance Methods

frameLength

Returns the number of bytes that the arguments, taken together, occupy on the stack.

```
- (NSUInteger)frameLength
```

Return Value
The number of bytes that the arguments, taken together, occupy on the stack.

Discussion
This number varies with the hardware architecture the application runs on.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSMethodSignature.h

getArgumentTypeAtIndex:
Returns the type encoding for the argument at a given index.

```
- (const char *)getArgumentTypeAtIndex:(NSUInteger)index
```

Parameters
index

The index of the argument to get.

Return Value
The type encoding for the argument at *index*.

Discussion
Indices begin with 0. The hidden arguments *self* (of type id) and *_cmd* (of type SEL) are at indices 0 and 1; method-specific arguments begin at index 2. Raises NSInvalidArgumentException if *index* is too large for the actual number of arguments.

Argument types are given as C strings with Objective-C type encoding. This encoding is implementation-specific, so applications should use it with caution.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSMethodSignature.h

isOneway
Returns a Boolean value that indicates whether the receiver is asynchronous when invoked through distributed objects.

```
- (BOOL)isOneway
```

Return Value
YES if the receiver is asynchronous when invoked through distributed objects, otherwise NO.

Discussion
If the method is oneway, the sender of the remote message doesn't block awaiting a reply.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSMethodSignature.h

methodReturnLength

Returns the number of bytes required for the return value.

```
- (NSUInteger)methodReturnLength
```

Return Value
The number of bytes required for the return value.

Availability
Available in iPhone OS 2.0 and later.

See Also
- methodReturnType (page 564)

Declared In
NSMethodSignature.h

methodReturnType

Returns a C string encoding the return type of the method in Objective-C type encoding.

```
- (const char *)methodReturnType
```

Return Value
A C string encoding the return type of the method in Objective-C type encoding.

Discussion
This encoding is implementation-specific, so applications should use it with caution.

Availability
Available in iPhone OS 2.0 and later.

See Also
- methodReturnLength (page 564)

Declared In
NSMethodSignature.h

numberOfArguments

Returns the number of arguments recorded in the receiver.

```
- (NSUInteger)numberOfArguments
```

Return Value
The number of arguments recorded in the receiver.

Discussion

There are always at least `2` arguments, because an `NSMethodSignature` object includes the hidden arguments *`self`* and *`_cmd`*, which are the first two arguments passed to every method implementation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSMethodSignature.h`

NSMutableArray Class Reference

Inherits from:	NSArray : NSObject
Conforms to:	NSCoding (NSArray) NSCopying (NSArray) NSMutableCopying (NSArray) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSArray.h Foundation/NSPredicate.h Foundation/NSSortDescriptor.h
Companion guides:	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableArray` class declares the programmatic interface to objects that manage a modifiable array of objects. This class adds insertion and deletion operations to the basic array-handling behavior inherited from `NSArray`.

NSArray and NSMutableArray are part of a class cluster, so arrays are not actual instances of the NSArray or NSMutableArray classes but of one of their private subclasses. Although an array's class is private, its interface is public, as declared by these abstract superclasses, NSArray and NSMutableArray. NSMutableArray's methods are conceptually based on these primitive methods:

insertObject:atIndex: (page 572)
removeObjectAtIndex: (page 577)
addObject: (page 571)
removeLastObject (page 575)
replaceObjectAtIndex:withObject: (page 581)

In a subclass, you must override all these methods, although you can implement the required functionality using just the first two (however this is likely to be inefficient).

The other methods in NSMutableArray's interface provide convenient ways of inserting an object into a specific slot in the array and removing an object based on its identity or position in the array.

Like NSArray, instances of NSMutableArray maintain strong references to their contents. If you do not use garbage collection, when you add an object to an array, the object receives a retain (page 1312) message. When an object is removed from a mutable array, it receives a release (page 1310) message. If there are no further references to the object, this means that the object is deallocated. If your program keeps a reference to such an object, the reference will become invalid unless you send the object a retain (page 1312) message before it's removed from the array. For example, if anObject is not retained before it is removed from the array, the third statement below could result in a runtime error:

```
id anObject = [[anArray objectAtIndex:0] retain];
[anArray removeObjectAtIndex:0];
[anObject someMessage];
```

NSMutableArray's filterUsingPredicate: provides in-place in-memory filtering of an array using a predicate. If you use the Core Data framework, this provides an efficient means of filtering an existing array of objects without—as a fetch does—requiring a round trip to a persistent data store.

Tasks

Creating and Initializing a Mutable Array

+ arrayWithCapacity: (page 570)
Creates and returns an NSMutableArray object with enough allocated memory to initially hold a given number of objects.

- initWithCapacity: (page 572)
Returns an array, initialized with enough memory to initially hold a given number of objects.

Adding Objects

- addObject: (page 571)
Inserts a given object at the end of the receiver.

- addObjectsFromArray: (page 571)
  Adds the objects contained in another given array to the end of the receiver's content.
- insertObject:atIndex: (page 572)
  Inserts a given object into the receiver's contents at a given index.
- insertObjects:atIndexes: (page 573)
  Inserts the objects in in a given array into the receiver at the specified indexes.

Removing Objects

- removeAllObjects (page 575)
  Empties the receiver of all its elements.
- removeLastObject (page 575)
  Removes the object with the highest-valued index in the receiver
- removeObject: (page 575)
  Removes all occurrences in the receiver of a given object.
- removeObject:inRange: (page 576)
  Removes all occurrences within a specified range in the receiver of a given object.
- removeObjectAtIndex: (page 577)
  Removes the object at *index*.
- removeObjectsAtIndexes: (page 578)
  Removes the objects at the specified indexes from the receiver.
- removeObjectIdenticalTo: (page 577)
  Removes all occurrences of a given object in the receiver.
- removeObjectIdenticalTo:inRange: (page 578)
  Removes all occurrences of *anObject* within the specified range in the receiver.
- removeObjectsFromIndices:numIndices: (page 579)
  Removes the specified number of objects from the receiver, beginning at the specified index.
- removeObjectsInArray: (page 580)
  Removes from the receiver the objects in another given array.
- removeObjectsInRange: (page 580)
  Removes from the receiver each of the objects within a given range.

Replacing Objects

- replaceObjectAtIndex:withObject: (page 581)
  Replaces the object at *index* with *anObject*.
- replaceObjectsAtIndexes:withObjects: (page 581)
  Replaces the objects in the receiver at specified locations specified with the objects from a given array.
- replaceObjectsInRange:withObjectsFromArray:range: (page 582)
  Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.

- `replaceObjectsInRange:withObjectsFromArray:` (page 582)
  Replaces the objects in the receiver specified by a given range with all of the objects from a given array.
- `setArray:` (page 583)
  Sets the receiver's elements to those in another given array.

Rearranging Content

- `exchangeObjectAtIndex:withObjectAtIndex:` (page 571)
  Exchanges the objects in the receiver at given indices.
- `sortUsingDescriptors:` (page 583)
  Sorts the receiver using a given array of sort descriptors.
- `sortUsingFunction:context:` (page 584)
  Sorts the receiver's elements in ascending order as defined by the comparison function *`compare`*.
- `sortUsingSelector:` (page 584)
  Sorts the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

Class Methods

arrayWithCapacity:

Creates and returns an `NSMutableArray` object with enough allocated memory to initially hold a given number of objects.

```
+ (id)arrayWithCapacity:(NSUInteger)numItems
```

Parameters

`numItems`
  The initial capacity of the new array.

Return Value

A new `NSMutableArray` object with enough allocated memory to hold *`numItems`* objects.

Discussion

Mutable arrays expand as needed; *`numItems`* simply establishes the object's initial capacity.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithCapacity:` (page 572)

Declared In

`NSArray.h`

Instance Methods

addObject:

Inserts a given object at the end of the receiver.

```
- (void)addObject:(id)anObject
```

Parameters

anObject

The object to add to the end of the receiver's content. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *anObject* is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addObjectsFromArray:` (page 571)
- `removeObject:` (page 575)
- `setArray:` (page 583)

Declared In

`NSArray.h`

addObjectsFromArray:

Adds the objects contained in another given array to the end of the receiver's content.

```
- (void)addObjectsFromArray:(NSArray *)otherArray
```

Parameters

otherArray

An array of objects to add to the end of the receiver's content.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setArray:` (page 583)
- `removeObject:` (page 575)

Declared In

`NSArray.h`

exchangeObjectAtIndex:withObjectAtIndex:

Exchanges the objects in the receiver at given indices.

```
- (void)exchangeObjectAtIndex:(NSUInteger)idx1 withObjectAtIndex:(NSUInteger)idx2
```

Parameters

idx1

The index of the object with which to replace the object at index *idx2*.

idx2

The index of the object with which to replace the object at index *idx1*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSArray.h

initWithCapacity:

Returns an array, initialized with enough memory to initially hold a given number of objects.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the new array.

Return Value

An array initialized with enough memory to hold *numItems* objects. The returned object might be different than the original receiver.

Discussion

Mutable arrays expand as needed; *numItems* simply establishes the object's initial capacity.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ arrayWithCapacity: (page 570)

Declared In

NSArray.h

insertObject:atIndex:

Inserts a given object into the receiver's contents at a given index.

```
- (void)insertObject:(id)anObject atIndex:(NSUInteger)index
```

Parameters

anObject

The object to add to the receiver's content. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *anObject* is `nil`.

index

The index in the receiver at which to insert *anObject*. This value must not be greater than the count of elements in the array.

Important: Raises an `NSRangeException` if *index* is greater than the number of elements in the array.

Discussion

If *index* is already occupied, the objects at *index* and beyond are shifted by adding `1` to their indices to make room.

Note that `NSArray` objects are not like C arrays. That is, even though you specify a size when you create an array, the specified size is regarded as a "hint"; the actual size of the array is still 0. This means that you cannot insert an object at an index greater than the current count of an array. For example, if an array contains two objects, its size is 2, so you can add objects at indices 0, 1, or 2. Index 3 is illegal and out of bounds; if you try to add an object at index 3 (when the size of the array is 2), `NSMutableArray` raises an exception.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- removeObjectAtIndex:` (page 577)

Declared In

`NSArray.h`

insertObjects:atIndexes:

Inserts the objects in in a given array into the receiver at the specified indexes.

```
- (void)insertObjects:(NSArray *)objects atIndexes:(NSIndexSet *)indexes
```

Parameters

objects

An array of objects to insert into the receiver.

indexes

The indexes at which the objects in *objects* should be inserted. The count of locations in *indexes* must equal the count of *objects*. For more details, see the Discussion.

Discussion

Each object in *objects* is inserted into the receiver in turn at the corresponding location specified in *indexes* after earlier insertions have been made. The implementation is conceptually similar to that illustrated in the following example.

```
- void insertObjects:(NSArray *additions) atIndexes:(NSIndexSet *indexes)
{
```

```
    int currentIndex = [indexes firstIndex];
    int i, count = [indexes count];

    for (i = 0; i < count; i++)
    {
        [self insertObject:[additions objectAtIndex:i] atIndex:currentIndex];
        currentIndex = [indexes indexGreaterThanIndex:currentIndex];
    }
}
```

The resulting behavior is illustrated by the following example.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:3];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, a, two, b, three, four)
```

The locations specified by *indexes* may therefore only exceed the bounds of the receiver if one location specifies the count of the array or the count of the array after preceding insertions, and other locations exceeding the bounds do so in a contiguous fashion from that location, as illustrated in the following examples.

In this example, both new objects are appended to the end of the array.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:5];
[indexes addIndex:4];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, two, three, four, a, b)
```

If you replace `[indexes addIndex:4]` with `[indexes addIndex:6]` (so that the indexes are 5 and 6), then the application will fail with an out of bounds exception.

In this example, two objects are added into the middle of the array, and another at the current end of the array (index 4) which means that it is third from the end of the modified array.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"two",
@"three", @"four", nil];
NSArray *newAdditions = [NSArray arrayWithObjects: @"a", @"b", @"c", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:2];
[indexes addIndex:4];
[array insertObjects:newAdditions atIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, a, b, two, c, three, four)
```

If you replace `[indexes addIndex:4]` with `[indexes addIndex:6]` (so that the indexes are 1, 2, and 6), then the output is `(one, a, b, two, three, four, c)`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 572)

Declared In
`NSArray.h`

removeAllObjects

Empties the receiver of all its elements.

```
- (void)removeAllObjects
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeObject:` (page 575)
- `removeLastObject` (page 575)
- `removeObjectAtIndex:` (page 577)
- `removeObjectIdenticalTo:` (page 577)

Declared In
`NSArray.h`

removeLastObject

Removes the object with the highest-valued index in the receiver

```
- (void)removeLastObject
```

Discussion
`removeLastObject` raises an `NSRangeException` if there are no objects in the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeAllObjects` (page 575)
- `removeObject:` (page 575)
- `removeObjectAtIndex:` (page 577)
- `removeObjectIdenticalTo:` (page 577)

Declared In
`NSArray.h`

removeObject:

Removes all occurrences in the receiver of a given object.

```
- (void)removeObject:(id)anObject
```

Parameters

anObject

The object to remove from the receiver.

Discussion

This method uses `indexOfObject:` (page 50) to locate matches and then removes them by using `removeObjectAtIndex:` (page 577). Thus, matches are determined on the basis of an object's response to the `isEqual:` message. If the receiver does not contain *anObject*, the method has no effect (although it does incur the overhead of searching the contents).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `removeAllObjects` (page 575)
- `removeLastObject` (page 575)
- `removeObjectAtIndex:` (page 577)
- `removeObjectIdenticalTo:` (page 577)
- `removeObjectsInArray:` (page 580)

Declared In

`NSArray.h`

removeObject:inRange:

Removes all occurrences within a specified range in the receiver of a given object.

```
- (void)removeObject:(id)anObject inRange:(NSRange)aRange
```

Parameters

anObject

The object to remove from the receiver's content.

aRange

The range from which to remove *anObject*.

Important: Raises an `NSRangeException` if *aRange* exceeds the bounds of the receiver.

Discussion

Matches are determined on the basis of an object's response to the `isEqual:` message. If the receiver does not contain *anObject* within *aRange*, the method has no effect (although it does incur the overhead of searching the contents).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `removeAllObjects` (page 575)
- `removeLastObject` (page 575)
- `removeObjectAtIndex:` (page 577)

- removeObjectIdenticalTo: (page 577)
- removeObjectsInArray: (page 580)

Declared In
NSArray.h

removeObjectAtIndex:

Removes the object at *index*.

```
- (void)removeObjectAtIndex:(NSUInteger)index
```

Parameters
index
The index from which to remove the object in the receiver. The value must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if *index* is beyond the end of the receiver.

Discussion
To fill the gap, all elements beyond *index* are moved by subtracting 1 from their index.

Availability
Available in iPhone OS 2.0 and later.

See Also
- insertObject:atIndex: (page 572)
- removeAllObjects (page 575)
- removeLastObject (page 575)
- removeObject: (page 575)
- removeObjectIdenticalTo: (page 577)
- removeObjectsFromIndices:numIndices: (page 579)

Declared In
NSArray.h

removeObjectIdenticalTo:

Removes all occurrences of a given object in the receiver.

```
- (void)removeObjectIdenticalTo:(id)anObject
```

Parameters
anObject
The object to remove from the receiver.

Discussion
This method uses the indexOfObjectIdenticalTo: (page 51) method to locate matches and then removes them by using removeObjectAtIndex: (page 577). Thus, matches are determined using object addresses. If the receiver does not contain *anObject*, the method has no effect (although it does incur the overhead of searching the contents).

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeAllObjects (page 575)
- removeLastObject (page 575)
- removeObject: (page 575)
- removeObjectAtIndex: (page 577)

Declared In
NSArray.h

removeObjectIdenticalTo:inRange:

Removes all occurrences of *anObject* within the specified range in the receiver.

```
- (void)removeObjectIdenticalTo:(id)anObject inRange:(NSRange)aRange
```

Parameters
anObject
The object to remove from the receiver within *aRange*.

aRange
The range in the receiver from which to remove *anObject*.

Important: Raises an NSRangeException if *aRange* exceeds the bounds of the receiver.

Discussion
This method uses the indexOfObjectIdenticalTo: (page 51) method to locate matches and then removes them by using removeObjectAtIndex: (page 577). Thus, matches are determined using object addresses. If the receiver does not contain *anObject* within *aRange*, the method has no effect (although it does incur the overhead of searching the contents).

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeAllObjects (page 575)
- removeLastObject (page 575)
- removeObject: (page 575)
- removeObjectAtIndex: (page 577)
- removeObjectsAtIndexes: (page 578)

Declared In
NSArray.h

removeObjectsAtIndexes:

Removes the objects at the specified indexes from the receiver.

```
- (void)removeObjectsAtIndexes:(NSIndexSet *)indexes
```

Parameters

indexes

The indexes of the objects to remove from the receiver. The locations specified by *indexes* must lie within the bounds of the receiver.

Discussion

This method is similar to `removeObjectAtIndex:` (page 577), but allows you to efficiently remove multiple objects with a single operation. *indexes* specifies the locations of objects to be removed given the state of the receiver when the method is invoked, as illustrated in the following example.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"a", @"two",
 @"b", @"three", @"four", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:3];
[array removeObjectsAtIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, two, three, four)
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithCapacity:` (page 572)
- `removeObjectAtIndex:` (page 577)
- `removeObject:inRange:` (page 576)

Declared In

`NSArray.h`

removeObjectsFromIndices:numIndices:

Removes the specified number of objects from the receiver, beginning at the specified index.

```
- (void)removeObjectsFromIndices:(NSUInteger *)indices numIndices:(NSUInteger)count
```

Parameters

indices

A C array of the indices of the objects to remove from the receiver.

count

The number of objects to remove from the receiver.

Discussion

This method is similar to `removeObjectAtIndex:` (page 577), but allows you to efficiently remove multiple objects with a single operation. If you sort the list of indices in ascending order, you will improve the speed of this operation.

This method cannot be sent to a remote object with distributed objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithCapacity:` (page 572)

- removeObjectAtIndex: (page 577)
- removeObject:inRange: (page 576)
- removeObjectsAtIndexes: (page 578)

Declared In
NSArray.h

removeObjectsInArray:

Removes from the receiver the objects in another given array.

```
- (void)removeObjectsInArray:(NSArray *)otherArray
```

Parameters

otherArray
An array containing the objects to be removed from the receiver.

Discussion
This method is similar to removeObject: (page 575), but allows you to efficiently remove large sets of objects with a single operation. If the receiver does not contain objects in *otherArray*, the method has no effect (although it does incur the overhead of searching the contents).

This method assumes that all elements in *otherArray* respond to hash and isEqual:.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeAllObjects (page 575)
- removeObjectIdenticalTo: (page 577)
- removeObjectsAtIndexes: (page 578)

Declared In
NSArray.h

removeObjectsInRange:

Removes from the receiver each of the objects within a given range.

```
- (void)removeObjectsInRange:(NSRange)aRange
```

Parameters

aRange
The range of the objects to remove from the receiver.

Discussion
The objects are removed using removeObjectAtIndex: (page 577).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSArray.h

replaceObjectAtIndex:withObject:

Replaces the object at *index* with *anObject*.

```
- (void)replaceObjectAtIndex:(NSUInteger)index withObject:(id)anObject
```

Parameters

index

The index of the object to be replaced. This value must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *index* is beyond the end of the receiver.

anObject

The object with which to replace the object at index *index* in the receiver. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *anObject* is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `insertObject:atIndex:` (page 572)
- `removeObjectAtIndex:` (page 577)
- `removeObjectsAtIndexes:` (page 578)
- `replaceObjectsAtIndexes:withObjects:` (page 581)

Declared In

`NSArray.h`

replaceObjectsAtIndexes:withObjects:

Replaces the objects in the receiver at specified locations specified with the objects from a given array.

```
- (void)replaceObjectsAtIndexes:(NSIndexSet *)indexes withObjects:(NSArray *)objects
```

Parameters

indexes

The indexes of the objects to be replaced.

objects

The objects with which to replace the objects in the receiver at the indexes specified by *indexes*. The count of locations in *indexes* must equal the count of *objects*.

Discussion

The indexes in *indexes* are used in the same order as the objects in *objects*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `insertObject:atIndex:` (page 572)

- removeObjectAtIndex: (page 577)
- replaceObjectAtIndex:withObject: (page 581)

Declared In
NSArray.h

replaceObjectsInRange:withObjectsFromArray:

Replaces the objects in the receiver specified by a given range with all of the objects from a given array.

```
- (void)replaceObjectsInRange:(NSRange)aRange withObjectsFromArray:(NSArray
    *)otherArray
```

Parameters

aRange
The range of objects to replace in (or remove from) the receiver.

otherArray
The array of objects from which to select replacements for the objects in *aRange*.

Discussion
If *otherArray* has fewer objects than are specified by *aRange*, the extra objects in the receiver are removed. If *otherArray* has more objects than are specified by *aRange*, the extra objects from *otherArray* are inserted into the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- insertObject:atIndex: (page 572)
- removeObjectAtIndex: (page 577)
- replaceObjectAtIndex:withObject: (page 581)
- replaceObjectsAtIndexes:withObjects: (page 581)

Declared In
NSArray.h

replaceObjectsInRange:withObjectsFromArray:range:

Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.

```
- (void)replaceObjectsInRange:(NSRange)aRange withObjectsFromArray:(NSArray
    *)otherArray range:(NSRange)otherRange
```

Parameters

aRange
The range of objects to replace in (or remove from) the receiver.

otherArray
The array of objects from which to select replacements for the objects in *aRange*.

otherRange
The range of objects to select from *otherArray* as replacements for the objects in *aRange*.

Discussion
The lengths of *aRange* and *otherRange* don't have to be equal: if *aRange* is longer than *otherRange*, the extra objects in the receiver are removed; if *otherRange* is longer than *aRange*, the extra objects from *otherArray* are inserted into the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 572)
- `removeObjectAtIndex:` (page 577)
- `replaceObjectAtIndex:withObject:` (page 581)
- `replaceObjectsAtIndexes:withObjects:` (page 581)

Declared In
`NSArray.h`

setArray:

Sets the receiver's elements to those in another given array.

```
- (void)setArray:(NSArray *)otherArray
```

Parameters
otherArray
The array of objects with which to replace the receiver's content.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addObjectsFromArray:` (page 571)
- `insertObject:atIndex:` (page 572)

Declared In
`NSArray.h`

sortUsingDescriptors:

Sorts the receiver using a given array of sort descriptors.

```
- (void)sortUsingDescriptors:(NSArray *)sortDescriptors
```

Parameters
sortDescriptors
An array containing the `NSSortDescriptor` objects to use to sort the receiver's contents.

Discussion
See NSSortDescriptor for additional information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sortUsingFunction:context:` (page 584)
- `sortUsingSelector:` (page 584)
- `sortedArrayUsingDescriptors:` (page 61) (`NSArray`)

Declared In
`NSSortDescriptor.h`

sortUsingFunction:context:

Sorts the receiver's elements in ascending order as defined by the comparison function *compare*.

```
- (void)sortUsingFunction:(int (*)(id, id, void *))compare context:(void *)context
```

Parameters
compare

The comparison function to use to compare two elements at a time.

The function's parameters are two objects to compare and the context parameter, *context*. The function should return `NSOrderedAscending` if the first element is smaller than the second, `NSOrderedDescending` if the first element is larger than the second, and `NSOrderedSame` if the elements are equal.

context

The context argument to pass to the compare function.

Discussion
This approach allows the comparison to be based on some outside parameter, such as whether character sorting is case-sensitive or case-insensitive.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `sortUsingDescriptors:` (page 583)
- `sortUsingSelector:` (page 584)
- `sortedArrayUsingFunction:context:` (page 62) (`NSArray`)

Declared In
`NSArray.h`

sortUsingSelector:

Sorts the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.

```
- (void)sortUsingSelector:(SEL)comparator
```

Parameters

comparator

A selector that specifies the comparison method to use to compare elements in the receiver.

The *comparator* message is sent to each object in the receiver and has as its single argument another object in the array. The *comparator* method should return `NSOrderedAscending` if the receiver is smaller than the argument, `NSOrderedDescending` if the receiver is larger than the argument, and `NSOrderedSame` if they are equal.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `sortUsingDescriptors:` (page 583)
- `sortUsingFunction:context:` (page 584)
- `sortedArrayUsingSelector:` (page 64) (`NSArray`)

Declared In

`NSArray.h`

NSMutableCharacterSet Class Reference

Inherits from:	NSCharacterSet : NSObject
Conforms to:	NSCopying NSMutableCopying NSCoding (NSCharacterSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSCharacterSet.h
Companion guide:	String Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableCharacterSet` class declares the programmatic interface to objects that manage a modifiable set of Unicode characters. You can add or remove characters from a mutable character set as numeric values in `NSRange` structures or as character values in strings, combine character sets by union or intersection, and invert a character set.

Mutable character sets are less efficient to use than immutable character sets. If you don't need to change a character set after creating it, create an immutable copy with `copy` and use that.

`NSMutableCharacterSet` defines no primitive methods. Subclasses must implement all methods declared by this class in addition to the primitives of `NSCharacterSet`. They must also implement `mutableCopyWithZone:` (page 1300).

Tasks

Adding and Removing Characters

- `addCharactersInRange:` (page 588)
  Adds to the receiver the characters whose Unicode values are in a given range.
- `removeCharactersInRange:` (page 590)
  Removes from the receiver the characters whose Unicode values are in a given range.
- `addCharactersInString:` (page 589)
  Adds to the receiver the characters in a given string.
- `removeCharactersInString:` (page 591)
  Removes from the receiver the characters in a given string.

Combining Character Sets

- `formIntersectionWithCharacterSet:` (page 589)
  Modifies the receiver so it contains only characters that exist in both the receiver and *`otherSet`*.
- `formUnionWithCharacterSet:` (page 590)
  Modifies the receiver so it contains all characters that exist in either the receiver or *`otherSet`*.

Inverting a Character Set

- `invert` (page 590)
  Replaces all the characters in the receiver with all the characters it didn't previously contain.

Instance Methods

addCharactersInRange:

Adds to the receiver the characters whose Unicode values are in a given range.

```
- (void)addCharactersInRange:(NSRange)aRange
```

Parameters

`aRange`

The range of characters to add.

`aRange``.location` is the value of the first character to add; *`aRange`*`.location + `*`aRange`*`.length- 1` is the value of the last. If *`aRange`*`.length` is `0`, this method has no effect.

Discussion
This code excerpt adds to a character set the lowercase English alphabetic characters:

```
NSMutableCharacterSet *aCharacterSet = [[NSMutableCharacterSet alloc] init];
NSRange lcEnglishRange;

lcEnglishRange.location = (unsigned int)'a';
lcEnglishRange.length = 26;
[aCharacterSet addCharactersInRange:lcEnglishRange];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeCharactersInRange:` (page 590)
- `addCharactersInString:` (page 589)

Declared In
`NSCharacterSet.h`

addCharactersInString:
Adds to the receiver the characters in a given string.

```
- (void)addCharactersInString:(NSString *)aString
```

Parameters
aString
The characters to add to the receiver.

Discussion
This method has no effect if *aString* is empty.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeCharactersInString:` (page 591)
- `addCharactersInRange:` (page 588)

Declared In
`NSCharacterSet.h`

formIntersectionWithCharacterSet:
Modifies the receiver so it contains only characters that exist in both the receiver and *otherSet*.

```
- (void)formIntersectionWithCharacterSet:(NSCharacterSet *)otherSet
```

Parameters
otherSet
The character set with which to perform the intersection.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `formUnionWithCharacterSet:` (page 590)

Declared In
`NSCharacterSet.h`

formUnionWithCharacterSet:

Modifies the receiver so it contains all characters that exist in either the receiver or *otherSet*.

```
- (void)formUnionWithCharacterSet:(NSCharacterSet *)otherSet
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `formIntersectionWithCharacterSet:` (page 589)

Declared In
`NSCharacterSet.h`

invert

Replaces all the characters in the receiver with all the characters it didn't previously contain.

```
- (void)invert
```

Discussion
Inverting a mutable character set, whether by `invert` or by `invertedSet` (page 144), is much less efficient than inverting an immutable character set with `invertedSet`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `invertedSet` (page 144) (NSCharacterSet)

Declared In
`NSCharacterSet.h`

removeCharactersInRange:

Removes from the receiver the characters whose Unicode values are in a given range.

```
- (void)removeCharactersInRange:(NSRange)aRange
```

Parameters

aRange

The range of characters to remove.

`aRange.location` is the value of the first character to remove; `aRange.location + aRange.length- 1` is the value of the last. If `aRange.length` is `0`, this method has no effect.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addCharactersInRange:` (page 588)
- `removeCharactersInString:` (page 591)

Declared In

`NSCharacterSet.h`

removeCharactersInString:

Removes from the receiver the characters in a given string.

```
- (void)removeCharactersInString:(NSString *)aString
```

Parameters

aString

The characters to remove from the receiver.

Discussion

This method has no effect if *aString* is empty.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addCharactersInString:` (page 589)
- `removeCharactersInRange:` (page 590)

Declared In

`NSCharacterSet.h`

NSMutableData Class Reference

Inherits from:	NSData : NSObject
Conforms to:	NSCoding (NSData) NSCopying (NSData) NSMutableCopying (NSData) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guide:	Binary Data Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSMutableData` (and its superclass `NSData`) provide data objects, object-oriented wrappers for byte buffers. Data objects let simple allocated buffers (that is, data with no embedded pointers) take on the behavior of Foundation objects. They are typically used for data storage and are also useful in Distributed Objects applications, where data contained in data objects can be copied or moved between applications. `NSData` creates static data objects, and `NSMutableData` creates dynamic data objects. You can easily convert one type of data object to the other with the initializer that takes an `NSData` object or an `NSMutableData` object as an argument.

`NSMutableData` is "toll-free bridged" with its Core Foundation counterpart, `CFData`. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSMutableData *` parameter, you can pass a `CFDataRef`, and in a function where you see a `CFDataRef` parameter, you can pass an `NSMutableData` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Creating and Initializing an NSMutableData Object

`+ dataWithCapacity:` (page 595)
Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

`+ dataWithLength:` (page 595)
Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

`- initWithCapacity:` (page 597)
Returns an initialized `NSMutableData` object capable of holding the specified number of bytes.

`- initWithLength:` (page 598)
Initializes and returns an `NSMutableData` object containing a given number of zeroed bytes.

Adjusting Capacity

`- increaseLengthBy:` (page 597)
Increases the length of the receiver by a given number of bytes.

`- setLength:` (page 601)
Extends or truncates a mutable data object to a given length.

Accessing Data

`- mutableBytes` (page 598)
Returns a pointer to the receiver's data.

Adding Data

`- appendBytes:length:` (page 596)
Appends to the receiver a given number of bytes from a given buffer.

`- appendData:` (page 596)
Appends the content of another `NSData` object to the receiver.

Modifying Data

- `replaceBytesInRange:withBytes:` (page 599)
  Replaces with a given set of bytes a given range within the contents of the receiver.
- `replaceBytesInRange:withBytes:length:` (page 599)
  Replaces with a given set of bytes a given range within the contents of the receiver.
- `resetBytesInRange:` (page 600)
  Replaces with zeroes the contents of the receiver in a given range.
- `setData:` (page 600)
  Replaces the entire contents of the receiver with the contents of another data object.

Class Methods

dataWithCapacity:

Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

```
+ (id)dataWithCapacity:(NSUInteger)aNumItems
```

Parameters

aNumItems
  The number of bytes the new data object can initially contain.

Return Value

A new `NSMutableData` object capable of holding *aNumItems* bytes.

Discussion

This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to `0`.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `+ dataWithLength:` (page 595)
- `- initWithCapacity:` (page 597)
- `- initWithLength:` (page 598)

Declared In

`NSData.h`

dataWithLength:

Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

```
+ (id)dataWithLength:(NSUInteger)length
```

Parameters

length

The number of bytes the new data object initially contains.

Return Value

A new `NSMutableData` object of *length* bytes, filled with zeros.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dataWithCapacity:` (page 595)
`- initWithCapacity:` (page 597)
`- initWithLength:` (page 598)

Declared In

`NSData.h`

Instance Methods

appendBytes:length:

Appends to the receiver a given number of bytes from a given buffer.

```
- (void)appendBytes:(const void *)bytes length:(NSUInteger)length
```

Parameters

bytes

A buffer containing data to append to the receiver's content.

length

The number of bytes from *bytes* to append.

Discussion

A sample using this method can be found in Working With Mutable Binary Data.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- appendData:` (page 596)

Declared In

`NSData.h`

appendData:

Appends the content of another `NSData` object to the receiver.

```
- (void)appendData:(NSData *)otherData
```

Parameters

otherData

The data object whose content is to be appended to the contents of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- appendBytes:length: (page 596)

Declared In

NSData.h

increaseLengthBy:

Increases the length of the receiver by a given number of bytes.

```
- (void)increaseLengthBy:(NSUInteger)extraLength
```

Parameters

extraLength

The number of bytes by which to increase the receiver's length.

Discussion

The additional bytes are all set to 0.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setLength: (page 601)

Declared In

NSData.h

initWithCapacity:

Returns an initialized NSMutableData object capable of holding the specified number of bytes.

```
- (id)initWithCapacity:(NSUInteger)capacity
```

Parameters

capacity

The number of bytes the data object can initially contain.

Return Value

An initialized NSMutableData object capable of holding *capacity* bytes.

Discussion

This method doesn't necessarily allocate the requested memory right away. Mutable data objects allocate additional memory as needed, so *aNumItems* simply establishes the object's initial capacity. When it does allocate the initial memory, though, it allocates the specified amount. This method sets the length of the data object to 0.

If the capacity specified in *aNumItems* is greater than four memory pages in size, this method may round the amount of requested memory up to the nearest full page.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithCapacity: (page 595)
- initWithLength: (page 598)

Declared In
NSData.h

initWithLength:

Initializes and returns an NSMutableData object containing a given number of zeroed bytes.

```
- (id)initWithLength:(NSUInteger)length
```

Parameters

length
The number of bytes the object initially contains.

Return Value
An initialized NSMutableData object containing *length* zeroed bytes.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ dataWithCapacity: (page 595)
+ dataWithLength: (page 595)
- initWithCapacity: (page 597)

Declared In
NSData.h

mutableBytes

Returns a pointer to the receiver's data.

```
- (void *)mutableBytes
```

Return Value
A pointer to the receiver's data.

Discussion
If the length of the receiver's data is not zero, this function is guaranteed to return a pointer to the object's internal bytes. If the length of receiver's data *is* zero, this function may or may not return NULL dependent upon many factors related to how the object was created (moreover, in this case the method result might change between different releases).

A sample using this method can be found in Working With Mutable Binary Data.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSData.h

replaceBytesInRange:withBytes:

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)bytes
```

Parameters
range
The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

bytes
The data to insert into the receiver's contents.

Discussion
If the location of *range* isn't within the receiver's range of bytes, an `NSRangeException` is raised. The receiver is resized to accommodate the new bytes, if necessary.

A sample using this method is given in Working With Mutable Binary Data.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `replaceBytesInRange:withBytes:length:` (page 599)
- `resetBytesInRange:` (page 600)

Declared In
NSData.h

replaceBytesInRange:withBytes:length:

Replaces with a given set of bytes a given range within the contents of the receiver.

```
- (void)replaceBytesInRange:(NSRange)range withBytes:(const void *)replacementBytes
    length:(NSUInteger)replacementLength
```

Parameters
range
The range within the receiver's contents to replace with `bytes`. The range must not exceed the bounds of the receiver.

replacementBytes
The data to insert into the receiver's contents.

replacementLength
The number of bytes to take from *replacementBytes*.

Discussion
If the length of *range* is not equal to *replacementLength*, the receiver is resized to accommodate the new bytes. Any bytes past *range* in the receiver are shifted to accommodate the new bytes. You can therefore pass NULL for *replacementBytes* and 0 for *replacementLength* to delete bytes in the receiver in the range *range*. You can also replace a range (which might be zero-length) with more bytes than the length of the range, which has the effect of insertion (or "replace some and insert more").

Availability
Available in iPhone OS 2.0 and later.

See Also
- replaceBytesInRange:withBytes: (page 599)

Declared In
NSData.h

resetBytesInRange:
Replaces with zeroes the contents of the receiver in a given range.

```
- (void)resetBytesInRange:(NSRange)range
```

Parameters
range

The range within the contents of the receiver to be replaced by zeros. The range must not exceed the bounds of the receiver.

Discussion
If the location of *range* isn't within the receiver's range of bytes, an NSRangeException is raised. The receiver is resized to accommodate the new bytes, if necessary.

Availability
Available in iPhone OS 2.0 and later.

See Also
- replaceBytesInRange:withBytes: (page 599)

Declared In
NSData.h

setData:
Replaces the entire contents of the receiver with the contents of another data object.

```
- (void)setData:(NSData *)aData
```

Parameters
aData

The data object whose content replaces that of the receiver.

Discussion
As part of its implementation, this method calls replaceBytesInRange:withBytes: (page 599).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSData.h

setLength:

Extends or truncates a mutable data object to a given length.

```
- (void)setLength:(NSUInteger)length
```

Parameters

length

The new length for the receiver.

Discussion
If the mutable data object is extended, the additional bytes are filled with zeros.

Availability
Available in iPhone OS 2.0 and later.

See Also
- increaseLengthBy: (page 597)

Declared In
NSData.h

NSMutableDictionary Class Reference

Inherits from:	NSDictionary : NSObject
Conforms to:	NSCoding (NSDictionary) NSCopying (NSDictionary) NSMutableCopying (NSDictionary) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDictionary.h
Companion guide:	Collections Programming Topics for Cocoa

Class at a Glance

An NSDictionary object stores a mutable set of entries.

Principal Attributes

- A count of the number of entries in the dictionary
- The set of keys contained in the dictionary
- The objects that correspond to the keys in the dictionary

`dictionaryWithCapacity:` (page 605)
Returns an empty dictionary with enough allocated space to hold a specified number of objects.

Commonly Used Methods

`removeObjectForKey:` (page 607)
Removes the specified entry from the dictionary.

`removeObjectsForKeys:` (page 608)
Removes multiple entries from the dictionary.

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableDictionary` class declares the programmatic interface to objects that manage mutable associations of keys and values. With its two efficient primitive methods—`setObject:forKey:` (page 609) and `removeObjectForKey:` (page 607)—this class adds modification operations to the basic operations it inherits from `NSDictionary`.

The other methods declared here operate by invoking one or both of these primitives. The non-primitive methods provide convenient ways of adding or removing multiple entries at a time.

When an entry is removed from a mutable dictionary, the key and value objects that make up the entry receive `release` (page 1310) messages. If there are no further references to the objects, they're deallocated. Note that if your program keeps a reference to such an object, the reference will become invalid unless you remember to send the object a `retain` message before it's removed from the dictionary. For example, the third statement below would result in a runtime error if `anObject` was not retained before it was removed:

```
id anObject = [[aDictionary objectForKey:theKey] retain];

[aDictionary removeObjectForKey:theKey];
[anObject someMessage];
```

Tasks

Creating and Initializing a Mutable Dictionary

`+ dictionaryWithCapacity:` (page 605)
Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

`- initWithCapacity:` (page 607)
Initializes a newly allocated mutable dictionary, allocating enough memory to hold *numItems* entries.

Adding Entries to a Mutable Dictionary

`- setObject:forKey:` (page 609)
Adds a given key-value pair to the receiver.

`- setValue:forKey:` (page 609)
Adds a given key-value pair to the receiver.

`- addEntriesFromDictionary:` (page 606)
Adds to the receiver the entries from another dictionary.

`- setDictionary:` (page 608)
Sets the contents of the receiver to entries in a given dictionary.

Removing Entries From a Mutable Dictionary

`- removeObjectForKey:` (page 607)
Removes a given key and its associated value from the receiver.

`- removeAllObjects` (page 607)
Empties the receiver of its entries.

`- removeObjectsForKeys:` (page 608)
Removes from the receiver entries specified by elements in a given array.

Class Methods

dictionaryWithCapacity:

Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

```
+ (id)dictionaryWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
The initial capacity of the new dictionary.

Return Value
A new mutable dictionary with enough allocated memory to hold *numItems* entries.

Discussion
Mutable dictionaries allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability
Available in iPhone OS 2.0 and later.

See Also
dictionary (page 305) (NSDictionary)
dictionaryWithContentsOfFile: (page 305) (NSDictionary)
dictionaryWithContentsOfURL: (page 306): (NSDictionary)
dictionaryWithObject:forKey: (page 307) (NSDictionary)
dictionaryWithObjects:forKeys: (page 307): (NSDictionary)
dictionaryWithObjects:forKeys:count: (page 308) (NSDictionary)
dictionaryWithObjectsAndKeys: (page 309) (NSDictionary)
- initWithCapacity: (page 607)

Declared In
NSDictionary.h

Instance Methods

addEntriesFromDictionary:

Adds to the receiver the entries from another dictionary.

```
- (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary
```

Parameters
otherDictionary
The dictionary from which to add entries

Discussion
Each value object from *otherDictionary* is sent a retain (page 1312) message before being added to the receiver. In contrast, each key object is copied (using copyWithZone: (page 1250)—keys must conform to the NSCopying protocol), and the copy is added to the receiver.

If both dictionaries contain the same key, the receiver's previous value object for that key is sent a release message, and the new value object takes its place.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setObject:forKey: (page 609)

Declared In
NSDictionary.h

initWithCapacity:

Initializes a newly allocated mutable dictionary, allocating enough memory to hold *numItems* entries.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems

The initial capacity of the initialized dictionary.

Return Value

An initialized mutable dictionary, which might be different than the original receiver.

Discussion

Mutable dictionaries allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ dictionaryWithCapacity: (page 605)

Declared In

NSDictionary.h

removeAllObjects

Empties the receiver of its entries.

```
- (void)removeAllObjects
```

Discussion

Each key and corresponding value object is sent a release (page 1310) message.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeObjectForKey: (page 607)
- removeObjectsForKeys: (page 608)

Declared In

NSDictionary.h

removeObjectForKey:

Removes a given key and its associated value from the receiver.

```
- (void)removeObjectForKey:(id)aKey
```

Parameters

aKey

The key to remove.

Discussion
Does nothing if *aKey* does not exist.

For example, assume you have an archived dictionary that records the call letters and associated frequencies of radio stations. To remove an entry for a defunct station, you could write code similar to the following:

```
NSMutableDictionary *stations = nil;

stations = [[NSMutableDictionary alloc]
        initWithContentsOfFile: pathToArchive];
[stations removeObjectForKey:@"KIKT"];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeAllObjects` (page 607)
- `removeObjectsForKeys:` (page 608)

Declared In
`NSDictionary.h`

removeObjectsForKeys:

Removes from the receiver entries specified by elements in a given array.

```
- (void)removeObjectsForKeys:(NSArray *)keyArray
```

Parameters
keyArray
An array of objects specifying the keys to remove.

Discussion
If a key in *keyArray* does not exist, the entry is ignored.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeObjectForKey:` (page 607)
- `removeObjectForKey:` (page 607)

Declared In
`NSDictionary.h`

setDictionary:

Sets the contents of the receiver to entries in a given dictionary.

```
- (void)setDictionary:(NSDictionary *)otherDictionary
```

Parameters

otherDictionary

A dictionary containing the new entries.

Discussion

All entries are removed from the receiver (with `removeAllObjects` (page 607)), then each entry from *otherDictionary* added into the receiver.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDictionary.h`

setObject:forKey:

Adds a given key-value pair to the receiver.

```
- (void)setObject:(id)anObject forKey:(id)aKey
```

Parameters

anObject

The value for *key*. The object receives a `retain` message before being added to the receiver. This value must not be `nil`.

aKey

The key for *value*. The key is copied (using `copyWithZone:` (page 1250); keys must conform to the `NSCopying` protocol). The key must not be *nil*.

Discussion

Raises an `NSInvalidArgumentException` if *aKey* or *anObject* is `nil`. If you need to represent a nil value in the dictionary, use `NSNull`.

If *aKey* already exists in the receiver, the receiver's previous value object for that key is sent a `release` (page 1310) message and *anObject* takes its place.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- removeObjectForKey:` (page 607)

Declared In

`NSDictionary.h`

setValue:forKey:

Adds a given key-value pair to the receiver.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value for *key*.

key

The key for *value*. Note that when using key-value coding, the key must be a string (see Key-Value Coding Fundamentals).

Discussion

This method adds *value* and *key* to the receiver using `setObject:forKey:` (page 609), unless *value* is `nil` in which case the method instead attempts to remove *key* using `removeObjectForKey:` (page 607).

Availability

Available in iPhone OS 2.0 and later.

See Also

`valueForKey:` (page 328) (NSDictionary)

Declared In

`NSKeyValueCoding.h`

NSMutableIndexSet Class Reference

Inherits from:	NSIndexSet : NSObject
Conforms to:	NSCopying (NSIndexSet) NSMutableCopying (NSIndexSet) NSCoding (NSIndexSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSIndexSet.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableIndexSet` class represents a mutable collection of unique unsigned integers, known as **indexes** because of the way they are used. This collection is referred to as a **mutable index set**.

The values in a mutable index set are always sorted, so the order in which values are added is irrelevant.

You must not subclass the `NSMutableIndexSet` class.

Tasks

Adding Indexes

- `addIndex:` (page 612)
  Adds an index to the receiver.
- `addIndexes:` (page 613)
  Adds the indexes in an index set to the receiver.
- `addIndexesInRange:` (page 613)
  Adds the indexes in an index range to the receiver.

Removing Indexes

- `removeIndex:` (page 614)
  Removes an index from the receiver.
- `removeIndexes:` (page 614)
  Removes the indexes in an index set from the receiver.
- `removeAllIndexes` (page 613)
  Removes the receiver's indexes.
- `removeIndexesInRange:` (page 615)
  Removes the indexes in an index range from the receiver.

Shifting Index Groups

- `shiftIndexesStartingAtIndex:by:` (page 615)
  Shifts a group of indexes to the left or the right within the receiver.

Instance Methods

addIndex:

Adds an index to the receiver.

```
- (void)addIndex:(NSUInteger)index
```

Parameters

index
  Index to add.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `addIndexes:` (page 613)

- addIndexesInRange: (page 613)

Declared In
NSIndexSet.h

addIndexes:

Adds the indexes in an index set to the receiver.

```
- (void)addIndexes:(NSIndexSet *)indexSet
```

Parameters

indexSet
Index set to add.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addIndex: (page 612)
- addIndexesInRange: (page 613)

Declared In
NSIndexSet.h

addIndexesInRange:

Adds the indexes in an index range to the receiver.

```
- (void)addIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
Index range to add. Must include only indexes representable as unsigned integers.

Discussion
This method raises an NSRangeException when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addIndex: (page 612)
- addIndexes: (page 613)

Declared In
NSIndexSet.h

removeAllIndexes

Removes the receiver's indexes.

```
- (void)removeAllIndexes
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeIndex:` (page 614)
- `removeIndexes:` (page 614)
- `removeIndexesInRange:` (page 615)

Declared In
`NSIndexSet.h`

removeIndex:

Removes an index from the receiver.

```
- (void)removeIndex:(NSUInteger)index
```

Parameters
index
Index to remove.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeAllIndexes` (page 613)
- `removeIndexes:` (page 614)
- `removeIndexesInRange:` (page 615)

Declared In
`NSIndexSet.h`

removeIndexes:

Removes the indexes in an index set from the receiver.

```
- (void)removeIndexes:(NSIndexSet *)indexSet
```

Parameters
indexSet
Index set to remove.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeIndex:` (page 614)
- `removeAllIndexes` (page 613)
- `removeIndexesInRange:` (page 615)

Declared In
NSIndexSet.h

removeIndexesInRange:

Removes the indexes in an index range from the receiver.

```
- (void)removeIndexesInRange:(NSRange)indexRange
```

Parameters

indexRange
Index range to remove.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeIndex: (page 614)
- removeIndexes: (page 614)
- removeAllIndexes (page 613)

Declared In
NSIndexSet.h

shiftIndexesStartingAtIndex:by:

Shifts a group of indexes to the left or the right within the receiver.

```
- (void)shiftIndexesStartingAtIndex:(NSUInteger)startIndex by:(NSInteger)delta
```

Parameters

startIndex
Head of the group of indexes to shift.

delta
Amount and direction of the shift. Positive integers shift the indexes to the right. Negative integers shift the indexes to the left.

Discussion
The group of indexes shifted is made up by *startIndex* and the indexes that follow it in the receiver.

A left shift deletes the indexes in the range (*startIndex*-*delta*,*delta*) from the receiver.

A right shift inserts empty space in the range (*indexStart*,*delta*) in the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSIndexSet.h

NSMutableSet Class Reference

Inherits from:	NSSet : NSObject
Conforms to:	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSSet.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableSet` class declares the programmatic interface to an object that manages a mutable set of objects. `NSMutableSet` provides support for the mathematical concept of a set. A set, both in its mathematical sense and in the `NSMutableSet` implementation, is an unordered collection of distinct elements.

The `NSCountedSet` class, which is a concrete subclass of `NSMutableSet`, supports mutable sets that can contain multiple instances of the same element. The `NSSet` class supports creating and managing immutable sets.

You add objects to an `NSMutableSet` object with `addObject:` (page 619), which adds a single object to the set; `addObjectsFromArray:` (page 620), which adds all objects from a specified array to the set; or `unionSet:` (page 623), which adds all the objects from another set. You remove objects from an `NSMutableSet` object using any of the methods `intersectSet:` (page 621), `minusSet:` (page 621), `removeAllObjects` (page 622), or `removeObject:` (page 622).

When an object is added to a set, it receives a `retain` (page 1312) message. When an object is removed from a mutable set, it receives a `release` (page 1310) message. If there are no further references to the object, this means that the object is deallocated. If your program keeps a reference to such an object, the reference will become invalid unless you send the object a `retain` (page 1312) message before it's removed from the array. For example, if `anObject` is not retained before it is removed from the set, the third statement below could result in a runtime error:

```
id anObject = [[aSet anyObject] retain];
[aSet removeObject:anObject];
[anObject someMessage];
```

Tasks

Creating a Mutable Set

`+ setWithCapacity:` (page 619)
Creates and returns a mutable set with a given initial capacity.

`- initWithCapacity:` (page 620)
Returns an initialized mutable set with a given initial capacity.

Adding and Removing Entries

`- addObject:` (page 619)
Adds a given object to the receiver, if it is not already a member.

`- removeObject:` (page 622)
Removes a given object from the receiver.

`- removeAllObjects` (page 622)
Empties the receiver of all of its members.

`- addObjectsFromArray:` (page 620)
Adds to the receiver each object contained in a given array that is not already a member.

Combining and Recombining Sets

`- unionSet:` (page 623)
Adds to the receiver each object contained in another given set that is not already a member.

`- minusSet:` (page 621)
Removes from the receiver each object contained in another given set that is present in the receiver.

- `intersectSet:` (page 621)
  Removes from the receiver each object that isn't a member of another given set.
- `setSet:` (page 622)
  Empties the receiver, then adds to the receiver each object contained in another given set.

Class Methods

setWithCapacity:

Creates and returns a mutable set with a given initial capacity.

```
+ (id)setWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
The initial capacity of the new set.

Return Value

A mutable set with initial capacity to hold *numItems* members.

Discussion

Mutable sets allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithCapacity:` (page 620)
+ `set` (page 919) (`NSSet`)
+ `setWithObjects:count:` (page 921) (`NSSet`)

Declared In

`NSSet.h`

Instance Methods

addObject:

Adds a given object to the receiver, if it is not already a member.

```
- (void)addObject:(id)anObject
```

Parameters

anObject
The object to add to the receiver.

Discussion

If *anObject* is already present in the set, this method has no effect on either the set or *anObject*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addObjectsFromArray:` (page 620)
- `unionSet:` (page 623)

Declared In
`NSSet.h`

addObjectsFromArray:

Adds to the receiver each object contained in a given array that is not already a member.

```
- (void)addObjectsFromArray:(NSArray *)anArray
```

Parameters

anArray
An array of objects to add to the receiver.

Discussion
If a given element of the array is already present in the set, this method has no effect on either the set or the array element.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addObject:` (page 619)
- `unionSet:` (page 623)

Declared In
`NSSet.h`

initWithCapacity:

Returns an initialized mutable set with a given initial capacity.

```
- (id)initWithCapacity:(NSUInteger)numItems
```

Parameters

numItems
The initial capacity of the set.

Return Value
An initialized mutable set with initial capacity to hold *numItems* members. The returned object might be different than the original receiver.

Discussion
Mutable sets allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ setWithCapacity: (page 619)

Declared In
NSSet.h

intersectSet:

Removes from the receiver each object that isn't a member of another given set.

```
- (void)intersectSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to perform the intersection.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeObject: (page 622)
- removeAllObjects (page 622)
- minusSet: (page 621)

Declared In
NSSet.h

minusSet:

Removes from the receiver each object contained in another given set that is present in the receiver.

```
- (void)minusSet:(NSSet *)otherSet
```

Parameters

otherSet
The set of objects to remove from the receiver.

Discussion
If any member of *otherSet* isn't present in the receiving set, this method has no effect on either the receiver or the *otherSet* member.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeObject: (page 622)
- removeAllObjects (page 622)
- intersectSet: (page 621)

Declared In
NSSet.h

removeAllObjects

Empties the receiver of all of its members.

```
- (void)removeAllObjects
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeObject:` (page 622)
- `minusSet:` (page 621)
- `intersectSet:` (page 621)

Declared In
`NSSet.h`

removeObject:

Removes a given object from the receiver.

```
- (void)removeObject:(id)anObject
```

Parameters
anObject
The object to remove from the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeAllObjects` (page 622)
- `minusSet:` (page 621)
- `intersectSet:` (page 621)

Declared In
`NSSet.h`

setSet:

Empties the receiver, then adds to the receiver each object contained in another given set.

```
- (void)setSet:(NSSet *)otherSet
```

Parameters
otherSet
The set whose members replace the receiver's content.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSSet.h`

unionSet:

Adds to the receiver each object contained in another given set that is not already a member.

```
- (void)unionSet:(NSSet *)otherSet
```

Parameters

otherSet

The set of objects to add to the receiver.

Discussion

If any member of *otherSet* is already present in the receiver, this method has no effect on either the receiver or the *otherSet* member.

Availability

Available in iPhone OS 2.0 and later.

See Also

- addObject: (page 619)
- addObjectsFromArray: (page 620)

Declared In

NSSet.h

NSMutableString Class Reference

Inherits from:	NSString : NSObject
Conforms to:	NSCoding (NSString) NSCopying (NSString) NSMutableCopying (NSString) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSString.h
Companion guide:	String Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableString` class declares the programmatic interface to an object that manages a mutable string—that is, a string whose contents can be edited—that conceptually represents an array of Unicode characters. To construct and manage an immutable string—or a string that cannot be changed after it has been created—use an object of the `NSString` class.

The `NSMutableString` class adds one primitive method—`replaceCharactersInRange:withString:` (page 629)—to the basic string-handling behavior inherited from `NSString`. All other methods that modify a string work through this method. For

example, `insertString:atIndex:` (page 629) simply replaces the characters in a range of `0` length, while `deleteCharactersInRange:` (page 628) replaces the characters in a given range with no characters.

Tasks

Creating and Initializing a Mutable String

`+ stringWithCapacity:` (page 626)
Returns an empty `NSMutableString` object with initial storage for a given number of characters.

`- initWithCapacity:` (page 628)
Returns an `NSMutableString` object initialized with initial storage for a given number of characters,

Modifying a String

`- appendFormat:` (page 627)
Adds a constructed string to the receiver.

`- appendString:` (page 627)
Adds to the end of the receiver the characters of a given string.

`- deleteCharactersInRange:` (page 628)
Removes from the receiver the characters in a given range.

`- insertString:atIndex:` (page 629)
Inserts into the receiver the characters of a given string at a given location.

`- replaceCharactersInRange:withString:` (page 629)
Replaces the characters from *`aRange`* with those in *`aString`*.

`- replaceOccurrencesOfString:withString:options:range:` (page 630)
Replaces all occurrences of a given string in a given range with another given string, returning the number of replacements.

`- setString:` (page 631)
Replaces the characters of the receiver with those in a given string.

Class Methods

stringWithCapacity:

Returns an empty `NSMutableString` object with initial storage for a given number of characters.

```
+ (id)stringWithCapacity:(NSUInteger)capacity
```

Parameters

capacity

The number of characters the string is expected to initially contain.

Return Value

An empty `NSMutableString` object with initial storage for *capacity* characters.

Discussion

The number of characters indicated by *capacity* is simply a hint to increase the efficiency of data storage. The value does *not* limit the length of the string.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

Instance Methods

appendFormat:

Adds a constructed string to the receiver.

```
- (void)appendFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for more information. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Discussion

The appended string is formed using `NSString`'s `stringWithFormat:` (page 976) method with the arguments listed.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- appendString:` (page 627)

Declared In

`NSString.h`

appendString:

Adds to the end of the receiver the characters of a given string.

```
- (void)appendString:(NSString *)aString
```

Parameters

aString

The string to append to the receiver. *aString* must not be `nil`

Availability

Available in iPhone OS 2.0 and later.

See Also

- appendFormat: (page 627)

Declared In

NSString.h

deleteCharactersInRange:

Removes from the receiver the characters in a given range.

```
- (void)deleteCharactersInRange:(NSRange)aRange
```

Parameters

aRange

The range of characters to delete. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Discussion

This method treats the length of the string as a valid range value that returns an empty string.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSString.h

initWithCapacity:

Returns an `NSMutableString` object initialized with initial storage for a given number of characters,

```
- (id)initWithCapacity:(NSUInteger)capacity
```

Parameters

capacity

The number of characters the string is expected to initially contain.

Return Value

An initialized `NSMutableString` object with initial storage for *capacity* characters. The returned object might be different than the original receiver.

Discussion

The number of characters indicated by *capacity* is simply a hint to increase the efficiency of data storage. The value does *not* limit the length of the string.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

insertString:atIndex:

Inserts into the receiver the characters of a given string at a given location.

```
- (void)insertString:(NSString *)aString atIndex:(NSUInteger)anIndex
```

Parameters
aString
The string to insert into the receiver. *aString* must not be nil.

anIndex
The location at which *aString* is inserted. The location must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if *anIndex* lies beyond the end of the string.

Discussion
The new characters begin at *anIndex* and the existing characters from *anIndex* to the end are shifted by the length of *aString*.

This method treats the length of the string as a valid index value that returns an empty string.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

replaceCharactersInRange:withString:

Replaces the characters from *aRange* with those in *aString*.

```
- (void)replaceCharactersInRange:(NSRange)aRange withString:(NSString *)aString
```

Parameters
aRange
The range of characters to replace. *aRange* must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the end of the receiver.

aString
The string with which to replace the characters in aRange. *aString* must not be nil.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

replaceOccurrencesOfString:withString:options:range:

Replaces all occurrences of a given string in a given range with another given string, returning the number of replacements.

```
- (NSUInteger)replaceOccurrencesOfString:(NSString *)target withString:(NSString
    *)replacement options:(NSStringCompareOptions)opts range:(NSRange)searchRange
```

Parameters

target

The string to replace.

Important: Raises an NSInvalidArgumentException if *target* is nil.

replacement

The string with which to replace *target*.

Important: Raises an NSInvalidArgumentException if *replacement* is nil.

opts

A mask specifying search options. See *String Programming Guide for Cocoa* for details.

If *opts* is NSBackwardsSearch, the search is done from the end of the range. If *opts* is NSAnchoredSearch, only anchored (but potentially multiple) instances are replaced. NSLiteralSearch and NSCaseInsensitiveSearch also apply.

searchRange

The range of characters to replace. *aRange* must not exceed the bounds of the receiver. Specify *searchRange* as NSMakeRange(0, [receiver length]) to process the entire string.

Important: Raises an NSRangeException if any part of *searchRange* lies beyond the end of the receiver.

Return Value
The number of replacements made.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

setString:

Replaces the characters of the receiver with those in a given string.

```
- (void)setString:(NSString *)aString
```

Parameters

aString

The string with which to replace the receiver's content. *aString* must not be `nil`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

NSMutableURLRequest Class Reference

Inherits from:	NSURLRequest : NSObject
Conforms to:	NSCopying (NSURLRequest) NSMutableCopying (NSURLRequest) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLRequest.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSMutableURLRequest is a subclass of NSURLRequest provided to aid developers who may find it more convenient to mutate a single request object for a series of URL load requests instead of creating an immutable NSURLRequest for each load.

This programming model is supported by the following contract between NSMutableURLRequest and NSURLConnection: NSURLConnection makes a deep copy of each NSMutableURLRequest object passed to one of its initializers.

NSMutableURLRequest, like NSURLRequest, is designed to be extended to support additional protocols by adding categories that access protocol specific values from a property object using NSURLProtocol's `propertyForKey:inRequest:` (page 1156) and `setProperty:forKey:inRequest:` (page 1158) methods.

Tasks

Setting Request Properties

- `setCachePolicy:` (page 635)
  Sets the cache policy of the receiver.
- `setMainDocumentURL:` (page 637)
  Sets the main document URL for the receiver.
- `setTimeoutInterval:` (page 638)
  Sets the receiver's timeout interval, in seconds.
- `setURL:` (page 638)
  Sets the URL of the receiver

Setting HTTP Specific Properties

- `addValue:forHTTPHeaderField:` (page 634)
  Adds an HTTP header to the receiver's HTTP header dictionary.
- `setAllHTTPHeaderFields:` (page 635)
  Replaces the receiver's header fields with the passed values.
- `setHTTPBody:` (page 636)
  Sets the request body of the receiver to the specified data.
- `setHTTPBodyStream:` (page 636)
  Sets the request body of the receiver to the contents of a specified input stream.
- `setHTTPMethod:` (page 636)
  Sets the receiver's HTTP request method.
- `setHTTPShouldHandleCookies:` (page 637)
  Sets whether the receiver should use the default cookie handling for the request.
- `setValue:forHTTPHeaderField:` (page 638)
  Sets the specified HTTP header field.

Instance Methods

addValue:forHTTPHeaderField:

Adds an HTTP header to the receiver's HTTP header dictionary.

```
- (void)addValue:(NSString *)value forHTTPHeaderField:(NSString *)field
```

Parameters

value

The value for the header field.

field

The name of the header field. In keeping with the HTTP RFC, HTTP header field names are case-insensitive

Discussion

This method provides the ability to add values to header fields incrementally. If a value was previously set for the specified *field*, the supplied *value* is appended to the existing value using the appropriate field delimiter. In the case of HTTP, this is a comma.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setValue:forHTTPHeaderField: (page 638)

Declared In

NSURLRequest.h

setAllHTTPHeaderFields:

Replaces the receiver's header fields with the passed values.

```
- (void)setAllHTTPHeaderFields:(NSDictionary *)headerFields
```

Parameters

headerFields

A dictionary with the new header fields. HTTP header fields must be string values; therefore, each object and key in the *headerFields* dictionary must be a subclass of NSString. If either the key or value for a key-value pair is not a subclass of NSString, the key-value pair is skipped.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setValue:forHTTPHeaderField: (page 638)

Declared In

NSURLRequest.h

setCachePolicy:

Sets the cache policy of the receiver.

```
- (void)setCachePolicy:(NSURLRequestCachePolicy)policy
```

Parameters

policy

The new cache policy.

Availability

Available in iPhone OS 2.0 and later.

See Also
- cachePolicy (page 1166)

Declared In
NSURLRequest.h

setHTTPBody:

Sets the request body of the receiver to the specified data.

```
- (void)setHTTPBody:(NSData *)data
```

Parameters

data

The new request body for the receiver. This is sent as the message body of the request, as in an HTTP POST request.

Discussion
Setting the HTTP body data clears any input stream set by setHTTPBodyStream: (page 636). These values are mutually exclusive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

setHTTPBodyStream:

Sets the request body of the receiver to the contents of a specified input stream.

```
- (void)setHTTPBodyStream:(NSInputStream *)inputStream
```

Parameters

inputStream

The input stream that will be the request body of the receiver. The entire contents of the stream will be sent as the body, as in an HTTP POST request. The *inputStream* should be unopened and the receiver will take over as the stream's delegate.

Discussion
Setting a body stream clears any data set by setHTTPBody: (page 636). These values are mutually exclusive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

setHTTPMethod:

Sets the receiver's HTTP request method.

```
- (void)setHTTPMethod:(NSString *)method
```

Parameters

method

The new HTTP request method. The default HTTP method is “GET”.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLRequest.h

setHTTPShouldHandleCookies:

Sets whether the receiver should use the default cookie handling for the request.

```
- (void)setHTTPShouldHandleCookies:(BOOL)handleCookies
```

Parameters

handleCookies

YES if the receiver should use the default cookie handling for the request, NO otherwise. The default is YES.

Special Considerations

In Mac OS X v10.2 with Safari 1.0 the value set by this method is not respected by the framework.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLRequest.h

setMainDocumentURL:

Sets the main document URL for the receiver.

```
- (void)setMainDocumentURL:(NSURL *)theURL
```

Parameters

theURL

The new main document URL. Can be nil.

Discussion

The caller should set the main document URL to an appropriate main document, if known. For example, when loading a web page the URL of the HTML document for the top-level frame would be appropriate. This URL will be used for the “only from same domain as main document” cookie accept policy.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLRequest.h

setTimeoutInterval:

Sets the receiver's timeout interval, in seconds.

```
- (void)setTimeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

timeoutInterval

The timeout interval, in seconds. If during a connection attempt the request remains idle for longer than the timeout interval, the request is considered to have timed out. The default timeout interval is 60 seconds.

Availability

Available in iPhone OS 2.0 and later.

See Also

- timeoutInterval (page 1169)

Declared In

NSURLRequest.h

setURL:

Sets the URL of the receiver

```
- (void)setURL:(NSURL *)theURL
```

Parameters

theURL

The new URL.

Availability

Available in iPhone OS 2.0 and later.

See Also

- URL (page 1170)

Declared In

NSURLRequest.h

setValue:forHTTPHeaderField:

Sets the specified HTTP header field.

```
- (void)setValue:(NSString *)value forHTTPHeaderField:(NSString *)field
```

Parameters

value

The new value for the header field. Any existing value for the field is replaced by the new value.

field

The name of the header field to set. In keeping with the HTTP RFC, HTTP header field names are case-insensitive.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- addValue:forHTTPHeaderField:` (page 634)

Declared In
`NSURLRequest.h`

NSNetService Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNetServices.h
Companion guides:	Bonjour Overview NSNetServices and CFNetServices Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSNetService` class represents a network service that your application publishes or uses as a client. This class and the `NSNetServiceBrowser` class use multicast DNS to convey information about network services to and from your application. The API of `NSNetService` provides a convenient way to publish the services offered by your application and to resolve the socket address for a service.

The types of services you access using `NSNetService` are the same types that you access directly using BSD sockets. HTTP and FTP are two services commonly provided by systems. (For a list of common services and the ports used by those services, see the file `/etc/services`.) Applications can also define their own custom services to provide specific data to clients.

You can use the `NSNetService` class as either a publisher of a service or as a client of a service. If your application publishes a service, your code must acquire a port and prepare a socket to communicate with clients. Once your socket is ready, you use the `NSNetService` class to notify clients that your service is ready. If your application is the client of a network service, you can either create an `NSNetService` object directly (if you know the exact host and port information) or you can use an `NSNetServiceBrowser` object to browse for services.

To publish a service, you must initialize your `NSNetService` object with the service name, domain, type, and port information. All of this information must be valid for the socket created by your application. Once initialized, you call the `publish` (page 649) method to broadcast your service information out to the network.

When connecting to a service, you would normally use the `NSNetServiceBrowser` class to locate the service on the network and obtain the corresponding `NSNetService` object. Once you have the object, you proceed to call the `resolveWithTimeout:` (page 651) method to verify that the service is available and ready for your application. If it is, the `addresses` (page 645) method returns the socket information you can use to connect to the service.

The methods of `NSNetService` operate asynchronously so that your application is not impacted by the speed of the network. All information about a service is returned to your application through the `NSNetService` object's delegate. You must provide a delegate object to respond to messages and to handle errors appropriately.

Tasks

Creating Network Services

- `initWithDomain:type:name:` (page 647)
  Returns the receiver, initialized as a network service of a given type and sets the initial host information.
- `initWithDomain:type:name:port:` (page 648)
  Initializes the receiver as a network service of type *type* at the socket location specified by *domain*, *name*, and *port*.

Configuring Network Services

+ `dataFromTXTRecordDictionary:` (page 644)
  Returns an `NSData` object representing a TXT record formed from a given dictionary.
+ `dictionaryFromTXTRecordData:` (page 645)
  Returns a dictionary representing a TXT record given as an `NSData` object.
- `addresses` (page 645)
  Returns an array containing `NSData` objects, each of which contains a socket address for the service.
- `domain` (page 646)
  Returns the domain name of the service.

- `getInputStream:outputStream:` (page 646)
  Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.
- `hostName` (page 647)
  Returns the host name of the computer providing the service.
- `name` (page 649)
  Returns the name of the service.
- `type` (page 654)
  Returns the type of the service.
- `TXTRecordData` (page 654)
  Returns the TXT record for the receiver.
- `setTXTRecordData:` (page 652)
  Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.
- `delegate` (page 646)
  Returns the delegate for the receiver.
- `setDelegate:` (page 652)
  Sets the delegate for the receiver.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 652)
  Adds the service to the specified run loop.
- `removeFromRunLoop:forMode:` (page 650)
  Removes the service from the given run loop for a given mode.

Using Network Services

- `publish` (page 649)
  Attempts to advertise the receiver's on the network.
- `publishWithOptions:` (page 650)
  Attempts to advertise the receiver on the network, with the given options.
- `netServiceWillPublish:` (page 657) *delegate method*
  Notifies the delegate that the network is ready to publish the service.
- `netService:didNotPublish:` (page 654) *delegate method*
  Notifies the delegate that a service could not be published.
- `netServiceDidPublish:` (page 656) *delegate method*
  Notifies the delegate that a service was successfully published.
- `resolveWithTimeout:` (page 651)
  Starts a resolve process of a finite duration for the receiver.
- `netServiceWillResolve:` (page 657) *delegate method*
  Notifies the delegate that the network is ready to resolve the service.
- `netService:didNotResolve:` (page 655) *delegate method*
  Informs the delegate that an error occurred during resolution of a given service.

- `netServiceDidResolveAddress:` (page 656) *delegate method*
  Informs the delegate that the address for a given service was resolved.
- `port` (page 649)
  Provides the port of the receiver.
- `startMonitoring` (page 653)
  Starts the monitoring of TXT-record updates for the receiver.
- `netService:didUpdateTXTRecordData:` (page 655) *delegate method*
  Notifies the delegate that the TXT record for a given service has been updated.
- `stop` (page 653)
  Halts a currently running attempt to publish or resolve a service.
- `stopMonitoring` (page 653)
  Stops the monitoring of TXT-record updates for the receiver.
- `netServiceDidStop:` (page 656) *delegate method*
  Informs the delegate that a `publish` (page 649) or `resolveWithTimeout:` (page 651) request was stopped.

Deprecated

- `resolve` (page 651)
  Starts a resolve process for the receiver. (Deprecated. Use `resolveWithTimeout:` (page 651) instead.)

Class Methods

dataFromTXTRecordDictionary:

Returns an `NSData` object representing a TXT record formed from a given dictionary.

```
+ (NSData *)dataFromTXTRecordDictionary:(NSDictionary *)txtDictionary
```

Parameters

txtDictionary
A dictionary containing a TXT record.

Return Value

An `NSData` object representing TXT data formed from *txtDictionary*. Returns `nil` if *txtDictionary* cannot be represented as an `NSData`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `TXTRecordData` (page 654)
+ `dictionaryFromTXTRecordData:` (page 645)

Declared In

`NSNetServices.h`

dictionaryFromTXTRecordData:

Returns a dictionary representing a TXT record given as an `NSData` object.

```
+ (NSDictionary *)dictionaryFromTXTRecordData:(NSData *)txtData
```

Parameters

txtData

A data object encoding a TXT record.

Return Value

A dictionary representing *txtData*. The dictionary's keys are `NSString` objects using UTF8 encoding. The values associated with all the dictionary's keys are `NSData` objects that encapsulate strings or data.

Returns `nil` if *txtData* cannot be represented as an `NSDictionary` object.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `TXTRecordData` (page 654)
+ `dataFromTXTRecordDictionary:` (page 644)

Declared In

`NSNetServices.h`

Instance Methods

addresses

Returns an array containing `NSData` objects, each of which contains a socket address for the service.

```
- (NSArray *)addresses
```

Return Value

An array containing `NSData` objects, each of which contains a socket address for the service. Each `NSData` object in the returned array contains an appropriate `sockaddr` structure that you can use to connect to the socket. The exact type of this structure depends on the service to which you are connecting. If no addresses were resolved for the service, the returned array contains zero elements.

Discussion

It is possible for a single service to resolve to more than one address or not resolve to any addresses. A service might resolve to multiple addresses if the computer publishing the service is currently multihoming.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `resolve` (page 651)

Declared In
NSNetServices.h

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setDelegate: (page 652)

Declared In
NSNetServices.h

domain

Returns the domain name of the service.

```
- (NSString *)domain
```

Return Value
The domain name of the service.

This can be an explicit domain name or it can contain the generic local domain name, @"local." (note the trailing period, which indicates an absolute name).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSNetServices.h

getInputStream:outputStream:

Retrieves by reference the input and output streams for the receiver and returns a Boolean value that indicates whether they were retrieved successfully.

```
- (BOOL)getInputStream:(NSInputStream **)inputStream outputStream:(NSOutputStream
    **)outputStream
```

Parameters

inputStream
Upon return, the input stream for the receiver.

outputStream
Upon return, the output stream for the receiver.

Return Value
`YES` if the streams are created successfully, otherwise `NO`.

Discussion
After this method is called, no delegate callbacks are called by the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

hostName
Returns the host name of the computer providing the service.

```
- (NSString *)hostName
```

Return Value
The host name of the computer providing the service. Returns `nil` if a successful resolve has not occurred.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

initWithDomain:type:name:
Returns the receiver, initialized as a network service of a given type and sets the initial host information.

```
- (id)initWithDomain:(NSString *)domain type:(NSString *)type name:(NSString *)name
```

Parameters
domain

The domain for the service. For the local domain, use `@"local."` not `@""`.

type

The network service type.

type must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string `"_http._tcp."`. Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

name

The name of the service to resolve.

Return Value
The receiver, initialized as a network service named *name* of type *type* in the domain *domain*.

Discussion
This method is the appropriate initializer to use to resolve a service—to publish a service, use initWithDomain:type:name:port: (page 648).

If you know the values for *domain*, *type*, and *name* of the service you wish to connect to, you can create an NSNetService object using this initializer and call resolveWithTimeout: (page 651) on the result.

You cannot use this initializer to publish a service. This initializer passes an invalid port number to the designated initializer, which prevents the service from being registered. Calling publish (page 649) on an NSNetService object initialized with this method generates a call to your delegate's netService:didNotPublish: (page 654) method with an NSNetServicesBadArgumentError error.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithDomain:type:name:port: (page 648)

Declared In
NSNetServices.h

initWithDomain:type:name:port:

Initializes the receiver as a network service of type *type* at the socket location specified by *domain*, *name*, and *port*.

```
- (id)initWithDomain:(NSString *)domain type:(NSString *)type name:(NSString *)name
    port:(int)port
```

Parameters
domain

The domain for the service. For the local domain, use @"local." not @"".

It is generally preferred to use a NSNetServiceBrowser object to obtain the local registration domain in which to publish your service. To use this default domain, simply pass in an empty string (@"").

type

The network service type.

type must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, as opposed to hosts, prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string "_http._tcp.". Note that the period character at the end of the string, which indicates that the domain name is an absolute name, is required.

name

The name by which the service is identified to the network. The name must be unique.

port

The port on which the service is published.

port must be a port number acquired by your application for the service.

Discussion
You use this method to create a service that you wish to publish on the network. Although you can also use this method to create a service you wish to resolve on the network, it is generally more appropriate to use the initWithDomain:type:name: (page 647) method instead.

When publishing a service, you must provide valid arguments in order to advertise your service correctly. If the host computer has access to multiple registration domains, you must create separate `NSNetService` objects for each domain. If you attempt to publish in a domain for which you do not have registration authority, your request may be denied.

It is acceptable to use an empty string for the *domain* argument when publishing or browsing a service, but do not rely on this for resolution.

This method is the designated initializer.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithDomain:type:name:` (page 647)

Declared In
`NSNetServices.h`

name

Returns the name of the service.

```
- (NSString *)name
```

Return Value
The name of the service.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

port

Provides the port of the receiver.

```
- (NSInteger)port
```

Return Value
The receiver's port. `-1` when it has not been resolved.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

publish

Attempts to advertise the receiver's on the network.

```
- (void)publish
```

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

Availability
Available in iPhone OS 2.0 and later.

See Also
- stop (page 653)

Declared In
NSNetServices.h

publishWithOptions:

Attempts to advertise the receiver on the network, with the given options.

```
- (void)publishWithOptions:(NSNetServiceOptions)serviceOptions
```

Parameters
serviceOptions
Options for the receiver.

Discussion
This method returns immediately, with success or failure indicated by the callbacks to the delegate.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSNetServices.h

removeFromRunLoop:forMode:

Removes the service from the given run loop for a given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters
aRunLoop
The run loop from which to remove the receiver.

mode
The run loop mode from which to remove the receiver. Possible values for *mode* are discussed in the "Constants" section of NSRunLoop.

Discussion
You can use this method in conjunction with scheduleInRunLoop:forMode: (page 652) to transfer the service to a different run loop. Although it is possible to remove an NSNetService object completely from any run loop and then attempt actions on it, it is an error to do so.

Availability
Available in iPhone OS 2.0 and later.

See Also
- scheduleInRunLoop:forMode: (page 652)

Declared In
NSNetServices.h

resolve

Starts a resolve process for the receiver. (**Deprecated.** Use resolveWithTimeout: (page 651) instead.)

```
- (void)resolve
```

Discussion
Attempts to determine at least one address for the receiver. This method returns immediately, with success or failure indicated by the callbacks to the delegate.

In Mac OS X v10.4, this method calls resolveWithTimeout: (page 651) with a timeout value of 5.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addresses (page 645)
- stop (page 653)
- resolveWithTimeout: (page 651)

Declared In
NSNetServices.h

resolveWithTimeout:

Starts a resolve process of a finite duration for the receiver.

```
- (void)resolveWithTimeout:(NSTimeInterval)timeout
```

Parameters
timeout
The maximum number of seconds to attempt a resolve.

Discussion
If the resolve succeeds before the *timeout* period lapses, the receiver sends netServiceDidResolveAddress: (page 656) to the delegate. Otherwise, the receiver sends netService:didNotResolve: (page 655) to the delegate.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addresses (page 645)
- stop (page 653)

Declared In
NSNetServices.h

scheduleInRunLoop:forMode:

Adds the service to the specified run loop.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop

The run loop to which to add the receiver.

mode

The run loop mode to which to add the receiver. Possible values for *mode* are discussed in the "Constants" section of `NSRunLoop`.

Discussion

You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 650) to transfer a service to a different run loop. You should not attempt to run a service on multiple run loops.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- removeFromRunLoop:forMode:` (page 650)

Declared In

`NSNetServices.h`

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate

The delegate for the receiver.

Discussion

The delegate is not retained.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- delegate` (page 646)

Declared In

`NSNetServices.h`

setTXTRecordData:

Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.

```
- (BOOL)setTXTRecordData:(NSData *)recordData
```

Parameters

recordData

The TXT record for the receiver.

Return Value

YES if *recordData* is successfully set as the TXT record, otherwise NO.

Availability

Available in iPhone OS 2.0 and later.

See Also

- TXTRecordData (page 654)

Declared In

NSNetServices.h

startMonitoring

Starts the monitoring of TXT-record updates for the receiver.

```
- (void)startMonitoring
```

Discussion

The delegate must implement netService:didUpdateTXTRecordData: (page 655), which is called when the TXT record for the receiver is updated.

Availability

Available in iPhone OS 2.0 and later.

See Also

- stopMonitoring (page 653)

Declared In

NSNetServices.h

stop

Halts a currently running attempt to publish or resolve a service.

```
- (void)stop
```

Discussion

This method results in the sending of a netServiceDidStop: (page 656) message to the delegate.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSNetServices.h

stopMonitoring

Stops the monitoring of TXT-record updates for the receiver.

```
- (void)stopMonitoring
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- startMonitoring (page 653)

Declared In
NSNetServices.h

TXTRecordData

Returns the TXT record for the receiver.

```
- (NSData *)TXTRecordData
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- setTXTRecordData: (page 652)
+ dictionaryFromTXTRecordData: (page 645)
+ dataFromTXTRecordDictionary: (page 644)

Declared In
NSNetServices.h

type

Returns the type of the service.

```
- (NSString *)type
```

Return Value
The type of the service.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSNetServices.h

Delegate Methods

netService:didNotPublish:

Notifies the delegate that a service could not be published.

```
- (void)netService:(NSNetService *)sender didNotPublish:(NSDictionary *)errorDict
```

Parameters

sender

The service that could not be published.

errorDict

A dictionary containing information about the problem. The dictionary contains the keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain`.

Discussion

This method may be called long after a `netServiceWillPublish:` (page 657) message has been delivered to the delegate.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNetServices.h`

netService:didNotResolve:

Informs the delegate that an error occurred during resolution of a given service.

```
- (void)netService:(NSNetService *)sender didNotResolve:(NSDictionary *)errorDict
```

Parameters

sender

The service that did not resolve.

errorDict

A dictionary containing information about the problem. The dictionary contains the keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain`.

Discussion

Clients may try to resolve again upon receiving this error. For example, a DNS rotary may yield different IP addresses on different resolution requests.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNetServices.h`

netService:didUpdateTXTRecordData:

Notifies the delegate that the TXT record for a given service has been updated.

```
- (void)netService:(NSNetService *)sender didUpdateTXTRecordData:(NSData *)data
```

Parameters

sender

The service whose TXT record was updated.

data

The new TXT record.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `startMonitoring` (page 653)

Declared In
`NSNetServices.h`

netServiceDidPublish:

Notifies the delegate that a service was successfully published.

```
- (void)netServiceDidPublish:(NSNetService *)sender
```

Parameters

sender
The service that was published.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

netServiceDidResolveAddress:

Informs the delegate that the address for a given service was resolved.

```
- (void)netServiceDidResolveAddress:(NSNetService *)sender
```

Parameters

sender
The service that was resolved.

Discussion
The delegate can use the `addresses` (page 645) method to retrieve the service's address.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addresses` (page 645)

Declared In
`NSNetServices.h`

netServiceDidStop:

Informs the delegate that a `publish` (page 649) or `resolveWithTimeout:` (page 651) request was stopped.

```
- (void)netServiceDidStop:(NSNetService *)sender
```

Parameters

sender

The service that stopped.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `stop` (page 653)

Declared In

`NSNetServices.h`

netServiceWillPublish:

Notifies the delegate that the network is ready to publish the service.

```
- (void)netServiceWillPublish:(NSNetService *)sender
```

Parameters

sender

The service that is ready to publish.

Discussion

Publication of the service proceeds asynchronously and may still generate a call to the delegate's `netService:didNotPublish:` (page 654) method if an error occurs.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNetServices.h`

netServiceWillResolve:

Notifies the delegate that the network is ready to resolve the service.

```
- (void)netServiceWillResolve:(NSNetService *)sender
```

Parameters

sender

The service that the network is ready to resolve.

Discussion

Resolution of the service proceeds asynchronously and may still generate a call to the delegate's `netService:didNotResolve:` (page 655) method if an error occurs.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNetServices.h`

Constants

NSNetServices Errors

If an error occurs, the delegate error-handling methods return a dictionary with the following keys.

```
extern NSString *NSNetServicesErrorCode;
extern NSString *NSNetServicesErrorDomain;
```

Constants

`NSNetServicesErrorCode`

This key identifies the error that occurred during the most recent operation.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesErrorDomain`

This key identifies the originator of the error, which is either the `NSNetService` object or the mach network layer. For most errors, you should not need the value provided by this key.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

Declared In

`NSNetServices.h`

NSNetServicesError

These constants identify errors that can occur when accessing net services.

```
typedef enum {
    NSNetServicesUnknownError = -72000,
    NSNetServicesCollisionError = -72001,
    NSNetServicesNotFoundError    = -72002,
    NSNetServicesActivityInProgress = -72003,
    NSNetServicesBadArgumentError = -72004,
    NSNetServicesCancelledError = -72005,
    NSNetServicesInvalidError = -72006,
    NSNetServicesTimeoutError = -72007,
} NSNetServicesError;
```

Constants

`NSNetServicesUnknownError`

An unknown error occurred.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesCollisionError`

The service could not be published because the name is already in use. The name could be in use locally or on another system.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesNotFoundError`
The service could not be found on the network.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesActivityInProgress`
The net service cannot process the request at this time. No additional information about the network state is known.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesBadArgumentError`
An invalid argument was used when creating the `NSNetService` object.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesCancelledError`
The client canceled the action.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesInvalidError`
The net service was improperly configured.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

`NSNetServicesTimeoutError`
The net service has timed out.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

Declared In
`NSNetServices.h`

NSNetServiceOptions

These constants specify options for a network service.

```
enum {
   NSNetServiceNoAutoRename = 1 << 0
};
typedef NSUInteger NSNetServiceOptions;
```

Constants
`NSNetServiceNoAutoRename`
Specifies that the network service not rename itself in the event of a name collision.

Available in iPhone OS 2.0 and later.

Declared in `NSNetServices.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSNetServices.h

NSNetServiceBrowser Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNetServices.h
Companion guides:	Bonjour Overview NSNetServices and CFNetServices Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSNetServiceBrowser` class defines an interface for finding published services on a network using multicast DNS. An instance of NSNetServiceBrowser is known as a **network service browser**.

Services can range from standard services, such as HTTP and FTP, to custom services defined by other applications. You can use a network service browser in your code to obtain the list of accessible domains and then to obtain an `NSNetService` object for each discovered service. Each network service browser performs one search at a time, so if you want to perform multiple simultaneous searches, use multiple network service browsers.

A network service browser performs all searches asynchronously using the current run loop to execute the search in the background. Results from a search are returned through the associated delegate object, which your client application must provide. Searching proceeds in the background until the object receives a `stop` (page 667) message.

To use an `NSNetServiceBrowser` object to search for services, allocate it, initialize it, and assign a delegate. (If you wish, you can also use the `scheduleInRunLoop:forMode:` (page 664) and `removeFromRunLoop:forMode:` (page 664) methods to execute searches on a run loop other than the current one.) Once your object is ready, you begin by gathering the list of accessible domains using either the `searchForRegistrationDomains` (page 665) or `searchForBrowsableDomains` (page 665) methods. From the list of returned domains, you can pick one and use the `searchForServicesOfType:inDomain:` (page 666) method to search for services in that domain.

The `NSNetServiceBrowser` class provides two ways to search for domains. In most cases, your client should use the `searchForRegistrationDomains` (page 665) method to search only for local domains to which the host machine has registration authority. This is the preferred method for accessing domains as it guarantees that the host machine can connect to services in the returned domains. Access to domains outside this list may be more limited.

Tasks

Creating Network Service Browsers

- `init` (page 663)
  Initializes an allocated `NSNetServiceBrowser` (page 661) object.

Configuring Network Service Browsers

- `delegate` (page 663)
  Returns the receiver's delegate.
- `setDelegate:` (page 666)
  Sets the receiver's delegate.

Using Network Service Browsers

- `searchForBrowsableDomains` (page 665)
  Initiates a search for domains visible to the host. This method returns immediately.
- `searchForRegistrationDomains` (page 665)
  Initiates a search for domains in which the host may register services.
- `netServiceBrowser:didFindDomain:moreComing:` (page 667) *delegate method*
  Tells the delegate the sender found a domain.
- `netServiceBrowser:didRemoveDomain:moreComing:` (page 669) *delegate method*
  Tells the delegate the a domain has disappeared or has become unavailable.
- `searchForServicesOfType:inDomain:` (page 666)
  Starts a search for services of a particular type within a specific domain.

- `netServiceBrowser:didFindService:moreComing:` (page 668) *delegate method*
  Tells the delegate the sender found a service.
- `netServiceBrowser:didRemoveService:moreComing:` (page 669) *delegate method*
  Tells the delegate a service has disappeared or has become unavailable.
- `netServiceBrowserWillSearch:` (page 670) *delegate method*
  Tells the delegate that a serch is commencing.
- `netServiceBrowser:didNotSearch:` (page 668) *delegate method*
  Tells the delegate that a search was not successful.
- `stop` (page 667)
  Halts a currently running search or resolution.
- `netServiceBrowserDidStopSearch:` (page 670) *delegate method*
  Tells the delegate that a search was stopped.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 664)
  Adds the receiver to the specified run loop.
- `removeFromRunLoop:forMode:` (page 664)
  Removes the receiver from the specified run loop.

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
Delegate for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDelegate:` (page 666)

Declared In
`NSNetServices.h`

init

Initializes an allocated `NSNetServiceBrowser` (page 661) object.

```
- (id)init
```

Return Value
Initialized `NSNetServiceBrowser` (page 661) object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

removeFromRunLoop:forMode:

Removes the receiver from the specified run loop.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)runLoopMode
```

Parameters

runLoop
Run loop from which to remove the receiver.

runLoopMode
Run loop mode in which to perform this operation, such as `NSDefaultRunLoopMode`. See the "Constants" (page 897) section of the `NSRunLoop` class for other run loop mode values.

Discussion
You can use this method in conjunction with `scheduleInRunLoop:forMode:` (page 664) to transfer the receiver to a run loop other than the default one. Although it is possible to remove an `NSNetService` object completely from any run loop and then attempt actions on it, you must not do it.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- scheduleInRunLoop:forMode:` (page 664)

Declared In
`NSNetServices.h`

scheduleInRunLoop:forMode:

Adds the receiver to the specified run loop.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)runLoopMode
```

Parameters

runLoop
Run loop from which to remove the receiver.

runLoopMode
Run loop mode in which to perform this operation, such as `NSDefaultRunLoopMode`. See the "Constants" (page 897) section of the `NSRunLoop` class for other run loop mode values.

Discussion
You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 664) to transfer the receiver to a run loop other than the default one. You should not attempt to run the receiver on multiple run loops.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeFromRunLoop:forMode:` (page 664)

Declared In
`NSNetServices.h`

searchForBrowsableDomains

Initiates a search for domains visible to the host. This method returns immediately.

```
- (void)searchForBrowsableDomains
```

Discussion
The delegate receives a `netServiceBrowser:didFindDomain:moreComing:` (page 667) message for each domain discovered.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `searchForRegistrationDomains` (page 665)

Declared In
`NSNetServices.h`

searchForRegistrationDomains

Initiates a search for domains in which the host may register services.

```
- (void)searchForRegistrationDomains
```

Discussion
This method returns immediately, sending a `netServiceBrowserWillSearch:` (page 670) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowser:didFindDomain:moreComing:` (page 667) message for each domain discovered.

Most network service browser clients do not have to use this method—it is sufficient to publish a service with the empty string, which registers it in any available registration domains automatically.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `searchForBrowsableDomains` (page 665)
- `searchForServicesOfType:inDomain:` (page 666)
- `netServiceBrowser:didFindDomain:moreComing:` (page 667)
- `netServiceBrowserWillSearch:` (page 670)

Declared In
`NSNetServices.h`

searchForServicesOfType:inDomain:

Starts a search for services of a particular type within a specific domain.

```
- (void)searchForServicesOfType:(NSString *)serviceType inDomain:(NSString
    *)domainName
```

Parameters

serviceType

Type of the service to search for.

domainName

Domain name in which to perform the search.

Discussion

This method returns immediately, sending a `netServiceBrowserWillSearch:` (page 670) message to the delegate if the network was ready to initiate the search.The delegate receives subsequent `netServiceBrowser:didFindService:moreComing:` (page 668) messages for each service discovered.

The *serviceType* argument must contain both the service type and transport layer information. To ensure that the mDNS responder searches for services, rather than hosts, make sure to prefix both the service name and transport layer name with an underscore character ("_"). For example, to search for an HTTP service on TCP, you would use the type string "`_http._tcp.`". Note that the period character at the end is required.

The *domainName* argument can be an explicit domain name, the generic local domain `@"local."` (note trailing period, which indicates an absolute name), or the empty string (`@""`), which indicates the default registration domain. Usually, you pass in an empty string. Note that it is acceptable to use an empty string for the *domainName* argument when publishing or browsing a service, but do not rely on this for resolution.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `netServiceBrowser:didFindService:moreComing:` (page 668)
- `netServiceBrowserWillSearch:` (page 670)

Declared In

`NSNetServices.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate

Object to serve as the receiver's delegate. Must not be `nil`.

Discussion

The delegate is not retained. The receiver calls the methods of your delegate to receive information about discovered domains and services.

Availability
Available in iPhone OS 2.0 and later.

See Also
- delegate (page 663)

Declared In
NSNetServices.h

stop

Halts a currently running search or resolution.

```
- (void)stop
```

Discussion
This method sends a netServiceBrowserDidStopSearch: (page 670) message to the delegate and causes the browser to discard any pending search results.

Availability
Available in iPhone OS 2.0 and later.

See Also
- netServiceBrowserDidStopSearch: (page 670)

Declared In
NSNetServices.h

Delegate Methods

netServiceBrowser:didFindDomain:moreComing:

Tells the delegate the sender found a domain.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didFindDomain:(NSString *)domainName moreComing:(BOOL)moreDomainsComing
```

Parameters

netServiceBrowser
Sender of this delegate message.

domainName
Name of the domain found by *netServiceBrowser*.

moreDomainsComing
YES when *netServiceBrowser* is waiting for additional domains. NO when there are no additional domains.

Discussion
The delegate uses this message to compile a list of available domains. It should wait until *moreDomainsComing* is NO to do a bulk update of user interface elements.

Availability
Available in iPhone OS 2.0 and later.

See Also
- searchForBrowsableDomains (page 665)
- searchForRegistrationDomains (page 665)

Declared In
NSNetServices.h

netServiceBrowser:didFindService:moreComing:

Tells the delegate the sender found a service.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didFindService:(NSNetService *)netService moreComing:(BOOL)moreServicesComing
```

Parameters
netServiceBrowser
Sender of this delegate message.

netService
Network service found by *netServiceBrowser*. The delegate can use this object to connect to and use the service.

moreServicesComing
YES when *netServiceBrowser* is waiting for additional services. NO when there are no additional services.

Discussion
The delegate uses this message to compile a list of available services. It should wait until *moreServicesComing* is NO to do a bulk update of user interface elements.

Special Considerations
If the delegate chooses to resolve *netService*, it should retain *netService* and set itself as that service's delegate. The delegate should, therefore, release that service when it receives the netServiceDidResolveAddress: (page 656) or netService:didNotResolve: (page 655) delegate messages of the NSNetService class.

Availability
Available in iPhone OS 2.0 and later.

See Also
- searchForServicesOfType:inDomain: (page 666)

Declared In
NSNetServices.h

netServiceBrowser:didNotSearch:

Tells the delegate that a search was not successful.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didNotSearch:(NSDictionary *)errorInfo
```

Parameters

netServiceBrowser
Sender of this delegate message.

errorInfo
Dictionary with the reasons the search was unsuccessful. Use the dictionary keys `NSNetServicesErrorCode` and `NSNetServicesErrorDomain` to retrieve the error information from the dictionary.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- netServiceBrowserWillSearch:` (page 670)

Declared In
`NSNetServices.h`

netServiceBrowser:didRemoveDomain:moreComing:

Tells the delegate the a domain has disappeared or has become unavailable.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didRemoveDomain:(NSString *)domainName moreComing:(BOOL)moreDomainsComing
```

Parameters

netServiceBrowser
Sender of this delegate message.

domainName
Name of the domain that became unavailable.

moreDomainsComing
`YES` when *netServiceBrowser* is waiting for additional domains. `NO` when there are no additional domains.

Discussion
The delegate uses this message to compile a list of unavailable domains. It should wait until *moreDomainsComing* is `NO` to do a bulk update of user interface elements.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNetServices.h`

netServiceBrowser:didRemoveService:moreComing:

Tells the delegate a service has disappeared or has become unavailable.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didRemoveService:(NSNetService *)netService moreComing:(BOOL)moreServicesComing
```

Parameters

netServiceBrowser
Sender of this delegate message.

netService
Network service that has become unavailable.

moreServicesComing
YES when *netServiceBrowser* is waiting for additional services. NO when there are no additional services.

Discussion

The delegate uses this message to compile a list of unavailable services. It should wait until *moreServicesComing* is NO to do a bulk update of user interface elements.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSNetServices.h

netServiceBrowserDidStopSearch:

Tells the delegate that a search was stopped.

```
- (void)netServiceBrowserDidStopSearch:(NSNetServiceBrowser *)netServiceBrowser
```

Parameters

netServiceBrowser
Sender of this delegate message.

Discussion

When *netServiceBrowser* receives a stop (page 667) message from its client, *netServiceBrowser* sends a netServiceBrowserDidStopSearch: message to its delegate. The delegate then performs any necessary cleanup.

Availability

Available in iPhone OS 2.0 and later.

See Also

- stop (page 667)

Declared In

NSNetServices.h

netServiceBrowserWillSearch:

Tells the delegate that a serch is commencing.

```
- (void)netServiceBrowserWillSearch:(NSNetServiceBrowser *)netServiceBrowser
```

Parameters

netServiceBrowser
Sender of this delegate message.

Discussion
This message is sent to the delegate only if the underlying network layer is ready to begin a search. The delegate can use this notification to prepare its data structures to receive data.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- netServiceBrowser:didNotSearch:` (page 668)

Declared In
`NSNetServices.h`

NSNotification Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNotification.h
Companion guide:	Notification Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSNotification objects encapsulate information so that it can be broadcast to other objects by an NSNotificationCenter object. An NSNotification object (referred to as a notification) contains a name, an object, and an optional dictionary. The name is a tag identifying the notification. The object is any object that the poster of the notification wants to send to observers of that notification (typically, it is the object that posted the notification). The dictionary stores other related objects, if any. NSNotification objects are immutable objects.

You can create a notification object with the class methods `notificationWithName:object:` (page 675) or `notificationWithName:object:userInfo:` (page 675). However, you don't usually create your own notifications directly. The NSNotificationCenter methods `postNotificationName:object:` (page 684) and `postNotificationName:object:userInfo:` (page 684) allow you to conveniently post a notification without creating it first.

NSCopying Protocol

The NSNotification class adopts the NSCopying protocol, making it possible to treat notifications as context-independent values that can be copied and reused. You can store a notification for later use or use the distributed objects system to send a notification to another process. The NSCopying protocol essentially allows clients to deal with notifications as first class values that can be copied by collections. You can put notifications in an array and send the `copy` message to that array, which recursively copies every item.

Creating Subclasses

You can subclass NSNotification to contain information in addition to the notification name, object, and dictionary. This extra data must be agreed upon between notifiers and observers.

NSNotification is a class cluster with no instance variables. As such, you must subclass NSNotification and override the primitive methods `name` (page 676), `object` (page 676), and `userInfo` (page 677). You can choose any designated initializer you like, but be sure that your initializer does not call NSNotification's implementation of `init` (via `[super init]`). NSNotification is not meant to be instantiated directly, and its `init` method raises an exception.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

Tasks

Creating Notifications

`+ notificationWithName:object:` (page 675)
Returns a new notification object with a specified name and object.

`+ notificationWithName:object:userInfo:` (page 675)
Returns a notification object with a specified name, object, and user information.

Getting Notification Information

- `name` (page 676)
  Returns the name of the notification.
- `object` (page 676)
  Returns the object associated with the notification.
- `userInfo` (page 677)
  Returns the user information dictionary associated with the receiver.

Class Methods

notificationWithName:object:

Returns a new notification object with a specified name and object.

```
+ (id)notificationWithName:(NSString *)aName object:(id)anObject
```

Parameters

aName
The name for the new notification. May not be `nil`.

anObject
The object for the new notification.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `postNotificationName:object:` (page 684) (NSNotificationCenter)

Declared In
`NSNotification.h`

notificationWithName:object:userInfo:

Returns a notification object with a specified name, object, and user information.

```
+ (id)notificationWithName:(NSString *)aName object:(id)anObject
    userInfo:(NSDictionary *)userInfo
```

Parameters

aName
The name for the new notification. May not be `nil`.

anObject
The object for the new notification.

userInfo
The user information dictionary for the new notification. May be `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also

+ notificationWithName:object: (page 675)
- postNotificationName:object:userInfo: (page 684) (NSNotificationCenter)

Declared In

NSNotification.h

Instance Methods

name

Returns the name of the notification.

```
- (NSString *)name
```

Return Value

The name of the notification. Typically you use this method to find out what kind of notification you are dealing with when you receive a notification.

Special Considerations

Notification names can be any string. To avoid name collisions, you might want to use a prefix that's specific to your application.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSNotification.h

object

Returns the object associated with the notification.

```
- (id)object
```

Return Value

The object associated with the notification. This is often the object that posted this notification. It may be nil.

Typically you use this method to find out what object a notification applies to when you receive a notification.

Discussion

For example, suppose you've registered an object to receive the message handlePortDeath: when the "PortInvalid" notification is posted to the notification center and that handlePortDeath: needs to access the object monitoring the port that is now invalid. handlePortDeath: can retrieve that object as shown here:

```
- (void)handlePortDeath:(NSNotification *)notification
{
    ...
```

```
    [self reclaimResourcesForPort:[notification object]];
    ...
}
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNotification.h`

userInfo

Returns the user information dictionary associated with the receiver.

```
- (NSDictionary *)userInfo
```

Return Value
Returns the user information dictionary associated with the receiver. May be `nil`.

The user information dictionary stores any additional objects that objects receiving the notification might use.

Discussion
For example, in the Application Kit, NSControl objects post the `NSControlTextDidChangeNotification` whenever the field editor (an NSText object) changes text inside the NSControl. This notification provides the NSControl object as the notification's associated object. In order to provide access to the field editor, the NSControl object posting the notification adds the field editor to the notification's user information dictionary. Objects receiving the notification can access the field editor and the NSControl object posting the notification as follows:

```
- (void)controlTextDidBeginEditing:(NSNotification *)notification
{
    NSText *fieldEditor = [[notification userInfo]
        objectForKey:@"NSFieldEditor"];               // the field editor
    NSControl *postingObject = [notification object]; // the object that posted
 the notification
    ...
}
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNotification.h`

NSNotificationCenter Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNotification.h
Companion guide:	Notification Programming Topics for Cocoa

Class at a Glance

The `NSNotificationCenter` class provides a way to send notifications to objects in the same task. It takes `NSNotification` objects and broadcasts them to any objects in the same task that have registered to receive the notification with the task's default notification center.

Principal Attributes

- **Notification dispatch table.** Each entry in this table specifies a notification set for a particular observer. A notification set is a subset of the notifications posted to the notification center. Each table entry contains three items:
  - **Notification observer:** Required. The object to be notified when qualifying notifications are posted to the notification center.
  - **Notification name:** Optional. Specifying a name reduces the set of notifications the entry specifies to those that have this name.
  - **Notification sender:** Optional. Specifying a sender reduces the set of notifications the entry specifies to those sent by this object.

  Table 53-1 shows the four types of dispatch table entries and the notification sets they specify. (This table omits the always present notification observer.)

Table 53-1 Types of dispatch table entries

Notification name	Notification sender	Notification set specified
Specified	Specified	Notifications with a particular name from a specific sender.
Specified	Unspecified	Notifications with a particular name by any sender.
Unspecified	Specified	Notifications posted by a specific sender.
Unspecified	Unspecified	All notifications.

Table 53-2 shows an example dispatch table with four observers.

Table 53-2 Example notification dispatch table

Observer	Notification name	Notification sender
`observerA`	`NSFileHandleReadCompletionNotification`	`nil`
`observerB`	`nil`	`addressTableView`
`observerC`	`NSWindowDidChangeScreenNotification`	`documentWindow`
`observerC`	`nil`	`addressTableView`
`observerD`	`nil`	`nil`

When notifications are posted to the notification center, each of the observers in Table 53-2 are notified of the following notifications:

- `observerA`: Notifications named `NSFileHandleReadCompletionNotification`.
- `observerB`: Notifications sent by `addressTableView`.
- `observerC`: Notifications named `NSWindowDidChangeScreenNotification` sent by `documentWindow` and notifications sent by `addressTableView`.
- `observerD`: All notifications.

Commonly Used Methods

`defaultCenter` (page 682)
Returns the task's default notification center.

`addObserver:selector:name:object:` (page 683)
Adds an entry to the notification center's dispatch table specifying at least an observer and a notification message.

`postNotificationName:object:` (page 684)
Creates and posts a notification to the notification center.

`removeObserver:` (page 685)
Removes all entries from the notification center's dispatch center that specify a particular observer, so that it no longer receives notifications posted to that notification center.

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSNotificationCenter` object (or simply, **notification center**) provides a mechanism for broadcasting information within a task. An `NSNotificationCenter` object is essentially a notification dispatch table.

Objects register with a notification center to receive notifications (`NSNotification` objects) using the `addObserver:selector:name:object:` (page 683) method. Each invocation of this method specifies a set of notifications. Therefore, objects may register as observers of different notification sets by calling `addObserver:selector:name:object:` several times.

When an object (known as the **notification sender**) posts a notification, it sends an `NSNotification` object to the notification center. The notification center then notifies any observers for which the notification meets the criteria specified on registration by sending them the specified notification message, passing the notification as the sole argument. The order in which observers receive notifications is undefined. It is possible for the posting object and the observing object to be the same.

A notification center delivers notifications to observers synchronously. In other words, the `postNotification:` (page 683) methods do not return until all observers have received and processed the notification. To send notifications asynchronously use `NSNotificationQueue`. In a multithreaded application, notifications are always delivered in the thread in which the notification was posted, which may not be the same thread in which an observer registered itself.

Important: The notification center does not retain its observers, therefore, you must ensure that you unregister observers (using `removeObserver:` (page 685) or `removeObserver:name:object:` (page 685)) before they are deallocated. (If you don't, you will generate a runtime error if the center sends a message to a freed object.)

Each task has a default notification center. You typically don't create your own. An `NSNotificationCenter` object can deliver notifications only within a single task. If you want to post a notification to other tasks or receive notifications from other tasks, use a `NSDistributedNotificationCenter` object.

Tasks

Getting the Notification Center

`+ defaultCenter` (page 682)
Returns the task's default notification center.

Managing Notification Observers

`- addObserver:selector:name:object:` (page 683)
Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.

`- removeObserver:` (page 685)
Removes all the entries specifying a given observer from the receiver's dispatch table.

`- removeObserver:name:object:` (page 685)
Removes matching entries from the receiver's dispatch table.

Posting Notifications

`- postNotification:` (page 683)
Posts a given notification to the receiver.

`- postNotificationName:object:` (page 684)
Creates a notification with a given name and sender and posts it to the receiver.

`- postNotificationName:object:userInfo:` (page 684)
Creates a notification with a given name, sender, and information and posts it to the receiver.

Class Methods

defaultCenter

Returns the task's default notification center.

```
+ (id)defaultCenter
```

Return Value

The current task's default notification center, which is used for system notifications.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotification.h`

Instance Methods

addObserver:selector:name:object:

Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.

```
- (void)addObserver:(id)notificationObserver selector:(SEL)notificationSelector
    name:(NSString *)notificationName object:(id)notificationSender
```

Parameters

notificationObserver

Object registering as an observer. Must not be `nil`.

notificationSelector

Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. The method the selector specifies must have one and only one argument.

notificationName

The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer. When `nil`, the notification center doesn't use a notification's name to decide whether to deliver it to the observer.

notificationSender

The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. When `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- removeObserver:` (page 685)

Declared In

`NSNotification.h`

postNotification:

Posts a given notification to the receiver.

```
- (void)postNotification:(NSNotification *)notification
```

Parameters

notification

The notification to post. This value must not be `nil`.

Discussion

You can create a notification with the `NSNotification` class method `notificationWithName:object:` (page 675) or `notificationWithName:object:userInfo:` (page 675). An exception is raised if *notification* is `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `postNotificationName:object:` (page 684)
- `postNotificationName:object:userInfo:` (page 684)

Declared In
`NSNotification.h`

postNotificationName:object:

Creates a notification with a given name and sender and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName
    object:(id)notificationSender
```

Parameters

notificationName
The name of the notification.

notificationSender
The object posting the notification.

Discussion
This method invokes `postNotificationName:object:userInfo:` (page 684) with a *userInfo* argument of `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `postNotification:` (page 683)

Declared In
`NSNotification.h`

postNotificationName:object:userInfo:

Creates a notification with a given name, sender, and information and posts it to the receiver.

```
- (void)postNotificationName:(NSString *)notificationName
    object:(id)notificationSender userInfo:(NSDictionary *)userInfo
```

Parameters

notificationName
The name of the notification.

notificationSender
The object posting the notification.

userInfo
Information about the the notification. May be `nil`.

Discussion
This method is the preferred method for posting notifications.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `postNotificationName:object:` (page 684)

Declared In
`NSNotification.h`

removeObserver:

Removes all the entries specifying a given observer from the receiver's dispatch table.

```
- (void)removeObserver:(id)notificationObserver
```

Parameters

notificationObserver
The observer to remove. Must not be `nil`.

Discussion
Be sure to invoke this method (or `removeObserver:name:object:` (page 685)) before *notificationObserver* or any object specified in `addObserver:selector:name:object:` (page 683) is deallocated.

The following example illustrates how to unregister `someObserver` for all notifications for which it had previously registered:

```
[[NSNotificationCenter defaultCenter] removeObserver:someObserver];
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNotification.h`

removeObserver:name:object:

Removes matching entries from the receiver's dispatch table.

```
- (void)removeObserver:(id)notificationObserver name:(NSString *)notificationName
    object:(id)notificationSender
```

Parameters

notificationObserver
Observer to remove from the dispatch table. Specify an observer to remove only entries for this observer. Must not be `nil`, or message will have no effect.

notificationName
Name of the notification to remove from dispatch table. Specify a notification name to remove only entries that specify this notification name. When `nil`, the receiver does not use notification names as criteria for removal.

notificationSender
Sender to remove from the dispatch table. Specify a notification sender to remove only entries that specify this sender. When `nil`, the receiver does not use notification senders as criteria for removal.

Discussion

Be sure to invoke this method (or `removeObserver:` (page 685)) before the observer object or any object specified in `addObserver:selector:name:object:` (page 683) is deallocated.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotification.h`

NSNotificationQueue Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNotificationQueue.h
Companion guide:	Notification Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSNotificationQueue objects (or simply notification queues) act as buffers for notification centers (instances of NSNotificationCenter). Whereas a notification center distributes notifications when posted, notifications placed into the queue can be delayed until the end of the current pass through the run loop or until the run loop is idle. Duplicate notifications can also be coalesced so that only one notification is sent although multiple notifications are posted. A notification queue maintains notifications (instances of NSNotification) generally in a first in first out (FIFO) order. When a notification rises to the front of the queue, the queue posts it to the notification center, which in turn dispatches the notification to all objects registered as observers.

Every thread has a default notification queue, which is associated with the default notification center for the task. You can create your own notification queues and have multiple queues per center and thread.

Tasks

Creating Notification Queues

`- initWithNotificationCenter:` (page 690)
Initializes and returns a notification queue for the specified notification center.

Getting the Default Queue

`+ defaultQueue` (page 688)
Returns the default notification queue for the current thread.

Managing Notifications

`- enqueueNotification:postingStyle:` (page 689)
Adds a notification to the notification queue with a specified posting style.

`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690)
Adds a notification to the notification queue with a specified posting style, criteria for coalescing, and runloop mode.

`- dequeueNotificationsMatching:coalesceMask:` (page 689)
Removes all notifications from the queue that match a provided notification using provided matching criteria.

Class Methods

defaultQueue

Returns the default notification queue for the current thread.

```
+ (NSNotificationQueue *)defaultQueue
```

Return Value

Returns the default notification queue for the current thread. This notification queue uses the default notification center.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

Instance Methods

dequeueNotificationsMatching:coalesceMask:

Removes all notifications from the queue that match a provided notification using provided matching criteria.

```
- (void)dequeueNotificationsMatching:(NSNotification *)notification
    coalesceMask:(NSUInteger)coalesceMask
```

Parameters

notification

The notification used for matching notifications to remove from the notification queue.

coalesceMask

A mask indicating what criteria to use when matching attributes of *notification* to attributes of notifications in the queue. The mask is created by combining any of the constants `NSNotificationNoCoalescing`, `NSNotificationCoalescingOnName`, and `NSNotificationCoalescingOnSender`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

enqueueNotification:postingStyle:

Adds a notification to the notification queue with a specified posting style.

```
- (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle
```

Parameters

notification

The notification to add to the queue.

postingStyle

The posting style for the notification. The posting style indicates when the notification queue should post the notification to its notification center.

Discussion

Notifications added with this method are posted using the runloop mode `NSDefaultRunLoopMode` and coalescing criteria that will coalesce only notifications that match both the notification's name and object.

This method invokes `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

enqueueNotification:postingStyle:coalesceMask:forModes:

Adds a notification to the notification queue with a specified posting style, criteria for coalescing, and runloop mode.

```
- (void)enqueueNotification:(NSNotification *)notification
    postingStyle:(NSPostingStyle)postingStyle coalesceMask:(NSUInteger)coalesceMask
     forModes:(NSArray *)modes
```

Parameters

notification

The notification to add to the queue.

postingStyle

The posting style for the notification. The posting style indicates when the notification queue should post the notification to its notification center.

coalesceMask

A mask indicating what criteria to use when matching attributes of *notification* to attributes of notifications in the queue. The mask is created by combining any of the constants `NSNotificationNoCoalescing`, `NSNotificationCoalescingOnName`, and `NSNotificationCoalescingOnSender`.

modes

The list of modes the notification may be posted in. The notification queue will only post the notification to its notification center if the run loops is in one of the modes provided in the array. May be `nil`, in which case it defaults to `NSDefaultRunLoopMode`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

initWithNotificationCenter:

Initializes and returns a notification queue for the specified notification center.

```
- (id)initWithNotificationCenter:(NSNotificationCenter *)notificationCenter
```

Parameters

notificationCenter

The notification center used by the new notification queue.

Return Value

The newly initialized notification queue.

Discussion

This is the designated initializer for the NSNotificationQueue class.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

Constants

NSNotificationCoalescing

These constants specify how notifications are coalesced.

```
typedef enum {
    NSNotificationNoCoalescing = 0,
    NSNotificationCoalescingOnName = 1,
    NSNotificationCoalescingOnSender = 2
} NSNotificationCoalescing;
```

Constants

`NSNotificationNoCoalescing`

Do not coalesce notifications in the queue.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

`NSNotificationCoalescingOnName`

Coalesce notifications with the same name.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

`NSNotificationCoalescingOnSender`

Coalesce notifications with the same object.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

Discussion

These constants are used in the third argument of `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690). You can OR them together to specify more than one.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

NSPostingStyle

These constants specify when notifications are posted.

```
typedef enum {
    NSPostWhenIdle = 1,
    NSPostASAP = 2,
    NSPostNow = 3
} NSPostingStyle;
```

Constants

`NSPostASAP`

The notification is posted at the end of the current notification callout or timer.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

`NSPostWhenIdle`

The notification is posted when the run loop is idle.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

`NSPostNow`

The notification is posted immediately after coalescing.

Available in iPhone OS 2.0 and later.

Declared in `NSNotificationQueue.h`

Discussion

These constants are used in both `enqueueNotification:postingStyle:` (page 689) and `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 690).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSNotificationQueue.h`

NSNull Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNull.h
Companion guide:	Number and Value Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSNull` class defines a singleton object used to represent null values in collection objects (which don't allow `nil` values).

Adopted Protocols

NSCoding

- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

Tasks

Obtaining an Instance

+ null (page 694)
Returns the singleton instance of NSNull.

Class Methods

null

Returns the singleton instance of NSNull.

```
+ (NSNull *)null
```

Return Value
The singleton instance of NSNull.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSNull.h

NSNumber Class Reference

Inherits from:	NSValue : NSObject
Conforms to:	NSCoding (NSValue) NSCopying (NSValue) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSValue.h Foundation/NSDecimalNumber.h
Companion guides:	Number and Value Programming Topics for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSNumber` is a subclass of `NSValue` that offers a value as any C scalar (numeric) type. It defines a set of methods specifically for setting and accessing the value as a signed or unsigned `char`, `short int`, `int`, `long int`, `long long int`, `float`, or `double` or as a `BOOL`. (Note that number objects do not necessarily preserve the type they are created with.) It also defines a `compare:` (page 705) method to determine the ordering of two `NSNumber` objects.

Creating a Subclass of NSNumber

As with any class cluster, if you create a subclass of `NSNumber`, you have to override the primitive methods of its superclass, `NSValue`. Furthermore, there is a restricted set of return values that your implementation of the `NSValue` method `objCType` can return, in order to take advantage of the abstract implementations of the non-primitive methods. The valid return values are "`c`", "`C`", "`s`", "`S`", "`i`", "`I`", "`l`", "`L`", "`q`", "`Q`", "`f`", and "`d`".

Tasks

Creating an NSNumber Object

`+ numberWithBool:` (page 699)
Creates and returns an `NSNumber` object containing a given value, treating it as a `BOOL`.

`+ numberWithChar:` (page 699)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `char`.

`+ numberWithDouble:` (page 700)
Creates and returns an `NSNumber` object containing a given value, treating it as a `double`.

`+ numberWithFloat:` (page 700)
Creates and returns an `NSNumber` object containing a given value, treating it as a `float`.

`+ numberWithInt:` (page 700)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `int`.

`+ numberWithInteger:` (page 701)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSInteger`.

`+ numberWithLong:` (page 701)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long`.

`+ numberWithLongLong:` (page 701)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long long`.

`+ numberWithShort:` (page 702)
Creates and returns an `NSNumber` object containing *`value`*, treating it as a signed `short`.

`+ numberWithUnsignedChar:` (page 702)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned char`.

`+ numberWithUnsignedInt:` (page 702)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned int`.

`+ numberWithUnsignedInteger:` (page 703)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSUInteger`.

`+ numberWithUnsignedLong:` (page 703)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long`.

+ numberWithUnsignedLongLong: (page 704)
    Creates and returns an NSNumber object containing a given value, treating it as an unsigned long long.
+ numberWithUnsignedShort: (page 704)
    Creates and returns an NSNumber object containing a given value, treating it as an unsigned short.

Initializing an NSNumber Object

- initWithBool: (page 707)
    Returns an NSNumber object initialized to contain a given value, treated as a BOOL.
- initWithChar: (page 708)
    Returns an NSNumber object initialized to contain a given value, treated as a signed char.
- initWithDouble: (page 708)
    Returns an NSNumber object initialized to contain *value*, treated as a double.
- initWithFloat: (page 709)
    Returns an NSNumber object initialized to contain a given value, treated as a float.
- initWithInt: (page 709)
    Returns an NSNumber object initialized to contain a given value, treated as a signed int.
- initWithInteger: (page 709)
    Returns an NSNumber object initialized to contain a given value, treated as an NSInteger.
- initWithLong: (page 710)
    Returns an NSNumber object initialized to contain a given value, treated as a signed long.
- initWithLongLong: (page 710)
    Returns an NSNumber object initialized to contain *value*, treated as a signed long long.
- initWithShort: (page 710)
    Returns an NSNumber object initialized to contain a given value, treated as a signed short.
- initWithUnsignedChar: (page 711)
    Returns an NSNumber object initialized to contain a given value, treated as an unsigned char.
- initWithUnsignedInt: (page 711)
    Returns an NSNumber object initialized to contain a given value, treated as an unsigned int.
- initWithUnsignedInteger: (page 711)
    Returns an NSNumber object initialized to contain a given value, treated as an NSUInteger.
- initWithUnsignedLong: (page 712)
    Returns an NSNumber object initialized to contain a given value, treated as an unsigned long.
- initWithUnsignedLongLong: (page 712)
    Returns an NSNumber object initialized to contain a given value, treated as an unsigned long long.
- initWithUnsignedShort: (page 712)
    Returns an NSNumber object initialized to contain a given value, treated as an unsigned short.

Accessing Numeric Values

- `boolValue` (page 704)
  Returns the receiver's value as a `BOOL`.
- `charValue` (page 705)
  Returns the receiver's value as a `char`.
- `decimalValue` (page 706)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 707)
  Returns the receiver's value as a `double`.
- `floatValue` (page 707)
  Returns the receiver's value as a `float`.
- `intValue` (page 713)
  Returns the receiver's value as an `int`.
- `integerValue` (page 713)
  Returns the receiver's value as an `NSInteger`.
- `longLongValue` (page 714)
  Returns the receiver's value as a `long long`.
- `longValue` (page 714)
  Returns the receiver's value as a `long`.
- `shortValue` (page 715)
  Returns the receiver's value as a `short`.
- `unsignedCharValue` (page 715)
  Returns the receiver's value as an unsigned `char`.
- `unsignedIntegerValue` (page 716)
  Returns the receiver's value as an `NSUInteger`.
- `unsignedIntValue` (page 716)
  Returns the receiver's value as an unsigned `int`.
- `unsignedLongLongValue` (page 716)
  Returns the receiver's value as an unsigned `long long`.
- `unsignedLongValue` (page 716)
  Returns the receiver's value as an unsigned `long`.
- `unsignedShortValue` (page 717)
  Returns the receiver's value as an unsigned `short`.

Retrieving String Representations

- `descriptionWithLocale:` (page 706)
  Returns a string that represents the contents of the receiver for a given locale.
- `stringValue` (page 715)
  Returns the receiver's value as a human-readable string.

Comparing NSNumber Objects

- `compare:` (page 705)
  Returns an `NSComparisonResult` value that indicates whether the receiver is greater than, equal to, or less than a given number.
- `isEqualToNumber:` (page 713)
  Returns a Boolean value that indicates whether the receiver and a given number are equal.

Accessing Type Information

- `objCType` (page 714)
  Returns a C string containing the Objective-C type of the data contained in the receiver.

Class Methods

numberWithBool:

Creates and returns an `NSNumber` object containing a given value, treating it as a `BOOL`.

```
+ (NSNumber *)numberWithBool:(BOOL)value
```

Parameters

value
  The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `BOOL`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

numberWithChar:

Creates and returns an `NSNumber` object containing a given value, treating it as a signed `char`.

```
+ (NSNumber *)numberWithChar:(char)value
```

Parameters

value
  The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `char`.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithDouble:

Creates and returns an NSNumber object containing a given value, treating it as a double.

```
+ (NSNumber *)numberWithDouble:(double)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a double.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithFloat:

Creates and returns an NSNumber object containing a given value, treating it as a float.

```
+ (NSNumber *)numberWithFloat:(float)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a float.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithInt:

Creates and returns an NSNumber object containing a given value, treating it as a signed int.

```
+ (NSNumber *)numberWithInt:(int)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed int.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithInteger:

Creates and returns an NSNumber object containing a given value, treating it as an NSInteger.

```
+ (NSNumber *)numberWithInteger:(NSInteger)value
```

Parameters

value
The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an NSInteger.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithLong:

Creates and returns an NSNumber object containing a given value, treating it as a signed long.

```
+ (NSNumber *)numberWithLong:(long)value
```

Parameters

value
The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed long.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

numberWithLongLong:

Creates and returns an NSNumber object containing a given value, treating it as a signed long long.

```
+ (NSNumber *)numberWithLongLong:(long long)value
```

Parameters

value
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as a signed `long long`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

numberWithShort:

Creates and returns an `NSNumber` object containing *`value`*, treating it as a signed `short`.

```
+ (NSNumber *)numberWithShort:(short)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as a signed `short`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

numberWithUnsignedChar:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned char`.

```
+ (NSNumber *)numberWithUnsignedChar:(unsigned char)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `unsigned char`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

numberWithUnsignedInt:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned int`.

```
+ (NSNumber *)numberWithUnsignedInt:(unsigned int)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned int`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

numberWithUnsignedInteger:

Creates and returns an `NSNumber` object containing a given value, treating it as an `NSUInteger`.

```
+ (NSNumber *)numberWithUnsignedInteger:(NSUInteger)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `NSUInteger`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

numberWithUnsignedLong:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long`.

```
+ (NSNumber *)numberWithUnsignedLong:(unsigned long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned long`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

numberWithUnsignedLongLong:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long long`.

```
+ (NSNumber *)numberWithUnsignedLongLong:(unsigned long long)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned long long`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

numberWithUnsignedShort:

Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned short`.

```
+ (NSNumber *)numberWithUnsignedShort:(unsigned short)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned short`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

Instance Methods

boolValue

Returns the receiver's value as a `BOOL`.

```
- (BOOL)boolValue
```

Return Value

The receiver's value as a `BOOL`, converting it as necessary.

Special Considerations

Prior to Mac OS X v10.3, the value returned isn't guaranteed to be one of `YES` or `NO`. A `0` value always means `NO` or false, but any nonzero value should be interpreted as `YES` or true.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

charValue

Returns the receiver's value as a `char`.

```
- (char)charValue
```

Return Value

The receiver's value as a `char`, converting it as necessary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

compare:

Returns an `NSComparisonResult` value that indicates whether the receiver is greater than, equal to, or less than a given number.

```
- (NSComparisonResult)compare:(NSNumber *)aNumber
```

Parameters

aNumber

The number with which to compare the receiver.

This value must not be `nil`. If the value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value

`NSOrderedAscending` if the value of *aNumber* is greater than the receiver's, `NSOrderedSame` if they're equal, and `NSOrderedDescending` if the value of *aNumber* is less than the receiver's.

Discussion

The `compare:` method follows the standard C rules for type conversion. For example, if you compare an `NSNumber` object that has an integer value with an `NSNumber` object that has a floating point value, the integer value is converted to a floating-point value for comparison.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

decimalValue

Returns the receiver's value, expressed as an `NSDecimal` structure.

```
- (NSDecimal)decimalValue
```

Return Value

The receiver's value, expressed as an `NSDecimal` structure. The value returned isn't guaranteed to be exact for `float` and `double` values.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

descriptionWithLocale:

Returns a string that represents the contents of the receiver for a given locale.

```
- (NSString *)descriptionWithLocale:(id)aLocale
```

Parameters

aLocale

An object containing locale information with which to format the description. Use `nil` if you don't want the description formatted.

Return Value

A string that represents the contents of the receiver formatted using the locale information in *locale*.

Discussion

For example, if you have an `NSNumber` object that has the integer value 522, sending it the `descriptionWithLocale:` message returns the string "522".

To obtain the string representation, this method invokes `NSString`'s `initWithFormat:locale:` (page 1006) method, supplying the format based on the type the `NSNumber` object was created with:

Data Type	Format Specification
`char`	%i
`double`	%0.16g
`float`	%0.7g
`int`	%i
`long`	%li
`long long`	%lli
`short`	%hi
`unsigned char`	%u

Data Type	Format Specification
`unsigned int`	%u
`unsigned long`	%lu
`unsigned long long`	%llu
`unsigned short`	%hu

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stringValue` (page 715)

Declared In
`NSValue.h`

doubleValue

Returns the receiver's value as a `double`.

```
- (double)doubleValue
```

Return Value
The receiver's value as a `double`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

floatValue

Returns the receiver's value as a `float`.

```
- (float)floatValue
```

Return Value
The receiver's value as a `float`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

initWithBool:

Returns an `NSNumber` object initialized to contain a given value, treated as a `BOOL`.

```
- (id)initWithBool:(BOOL)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `BOOL`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

initWithChar:

Returns an `NSNumber` object initialized to contain a given value, treated as a signed `char`.

```
- (id)initWithChar:(char)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `char`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

initWithDouble:

Returns an `NSNumber` object initialized to contain *value*, treated as a `double`.

```
- (id)initWithDouble:(double)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `double`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

initWithFloat:

Returns an `NSNumber` object initialized to contain a given value, treated as a `float`.

```
- (id)initWithFloat:(float)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a `float`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

initWithInt:

Returns an `NSNumber` object initialized to contain a given value, treated as a signed `int`.

```
- (id)initWithInt:(int)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as a signed `int`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

initWithInteger:

Returns an `NSNumber` object initialized to contain a given value, treated as an `NSInteger`.

```
- (id)initWithInteger:(NSInteger)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `NSInteger`.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithLong:

Returns an NSNumber object initialized to contain a given value, treated as a signed long.

```
- (id)initWithLong:(long)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed long.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithLongLong:

Returns an NSNumber object initialized to contain *value*, treated as a signed long long.

```
- (id)initWithLongLong:(long long)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed long long.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithShort:

Returns an NSNumber object initialized to contain a given value, treated as a signed short.

```
- (id)initWithShort:(short)value
```

Parameters

value

The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as a signed short.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithUnsignedChar:

Returns an NSNumber object initialized to contain a given value, treated as an unsigned char.

```
- (id)initWithUnsignedChar:(unsigned char)value
```

Parameters

value
The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an unsigned char.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithUnsignedInt:

Returns an NSNumber object initialized to contain a given value, treated as an unsigned int.

```
- (id)initWithUnsignedInt:(unsigned int)value
```

Parameters

value
The value for the new number.

Return Value
An NSNumber object containing *value*, treating it as an unsigned int.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

initWithUnsignedInteger:

Returns an NSNumber object initialized to contain a given value, treated as an NSUInteger.

```
- (id)initWithUnsignedInteger:(NSUInteger)value
```

Parameters

value
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `NSUInteger`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

initWithUnsignedLong:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long`.

```
- (id)initWithUnsignedLong:(unsigned long)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `unsigned long`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

initWithUnsignedLongLong:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long long`.

```
- (id)initWithUnsignedLongLong:(unsigned long long)value
```

Parameters

`value`
The value for the new number.

Return Value
An `NSNumber` object containing *`value`*, treating it as an `unsigned long long`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

initWithUnsignedShort:

Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned short`.

```
- (id)initWithUnsignedShort:(unsigned short)value
```

Parameters

value

The value for the new number.

Return Value

An `NSNumber` object containing *value*, treating it as an `unsigned short`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

integerValue

Returns the receiver's value as an `NSInteger`.

```
- (NSInteger)integerValue
```

Return Value

The receiver's value as an `NSInteger`, converting it as necessary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

intValue

Returns the receiver's value as an `int`.

```
- (int)intValue
```

Return Value

The receiver's value as an `int`, converting it as necessary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSValue.h`

isEqualToNumber:

Returns a Boolean value that indicates whether the receiver and a given number are equal.

```
- (BOOL)isEqualToNumber:(NSNumber *)aNumber
```

Parameters

aNumber

The number with which to compare the receiver.

Return Value
YES if the receiver and *aNumber* are equal, otherwise NOr

Discussion
Two NSNumber objects are considered equal if they have the same id values or if they have equivalent values (as determined by the compare: (page 705) method).

This method is more efficient than compare: (page 705) if you know the two objects are numbers.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

longLongValue

Returns the receiver's value as a long long.

```
- (long long)longLongValue
```

Return Value
The receiver's value as a long long, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

longValue

Returns the receiver's value as a long.

```
- (long)longValue
```

Return Value
The receiver's value as a long, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver.

```
- (const char *)objCType
```

Return Value
A C string containing the Objective-C type of the data contained in the receiver, as encoded by the `@encode()` compiler directive.

Special Considerations
The returned type does not necessarily match the method the receiver was created with.

shortValue
Returns the receiver's value as a `short`.

```
- (short)shortValue
```

Return Value
The receiver's value as a `short`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

stringValue
Returns the receiver's value as a human-readable string.

```
- (NSString *)stringValue
```

Return Value
The receiver's value as a human-readable string, created by invoking `descriptionWithLocale:` (page 706) where locale is `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

unsignedCharValue
Returns the receiver's value as an unsigned `char`.

```
- (unsigned char)unsignedCharValue
```

Return Value
The receiver's value as an unsigned `char`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

unsignedIntegerValue

Returns the receiver's value as an `NSUInteger`.

```
- (NSUInteger)unsignedIntegerValue
```

Return Value
The receiver's value as an `NSUInteger`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

unsignedIntValue

Returns the receiver's value as an unsigned `int`.

```
- (unsigned int)unsignedIntValue
```

Return Value
The receiver's value as an unsigned `int`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

unsignedLongLongValue

Returns the receiver's value as an unsigned `long long`.

```
- (unsigned long long)unsignedLongLongValue
```

Return Value
The receiver's value as an unsigned `long long`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSValue.h`

unsignedLongValue

Returns the receiver's value as an unsigned `long`.

```
- (unsigned long)unsignedLongValue
```

Return Value
The receiver's value as an unsigned `long`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

unsignedShortValue

Returns the receiver's value as an unsigned `short`.

```
- (unsigned short)unsignedShortValue
```

Return Value
The receiver's value as an unsigned `short`, converting it as necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

NSNumberFormatter Class Reference

Inherits from:	NSFormatter : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSNumberFormatter.h
Companion guide:	Data Formatting Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

Instances of `NSNumberFormatter` format the textual representation of cells that contain `NSNumber` objects and convert textual representations of numeric values into `NSNumber` objects. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. `NSNumberFormatter` objects can also impose ranges on the numeric values cells can accept.

Many new methods were added to `NSNumberFormatter` for Mac OS X v10.4 with the intent of making the class interface more like that of `CFNumberFormatter`, the Core Foundation service on which the class is based. The behavior of an `NSNumberFormatter` object can conform either to the range of behaviors existing prior to Mac OS X v10.4 or to the range of behavior since that release. (Methods

added for and since Mac OS X v10.4 are indicated by a method's availability statement.) You can determine the current formatter behavior with the `formatterBehavior` (page 731) method and you can set the formatter behavior with the `setFormatterBehavior:` (page 749) method.

Important: The pre-Mac OS X v10.4 methods of `NSNumberFormatter` are not compatible with the methods added for Mac OS X v10.4. An `NSNumberFormatter` object should not invoke methods in these different behavior groups indiscriminately. Use the old-style methods if you have configured the number-formatter behavior to be `NSNumberFormatterBehavior10_0`. Use the new methods instead of the older-style ones if you have configured the number-formatter behavior to be `NSNumberFormatterBehavior10_4`.

Note also that number formatters created in Interface Builder use the Mac OS X v10.0 behavior—see NSNumberFormatter on MacÂ OSÂ XÂ 10.4.

Nomenclature note: `NSNumberFormatter` provides several methods (such as `setMaximumFractionDigits:` (page 752)) that allow you to manage the number of **fraction digits** allowed as input by an instance: "fraction digits" are the numbers after the decimal separator (in English locales typically referred to as the "decimal point").

Tasks

Configuring Formatter Behavior and Style

- `setFormatterBehavior:` (page 749)
  Sets the formatter behavior of the receiver.
- `formatterBehavior` (page 731)
  Returns an `NSNumberFormatterBehavior` constant that indicates the formatter behavior of the receiver.
+ `setDefaultFormatterBehavior:` (page 727)
  Sets the default formatter behavior for new instances of `NSNumberFormatter` .
+ `defaultFormatterBehavior` (page 727)
  Returns an `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.
- `setNumberStyle:` (page 758)
  Sets the number style used by the receiver.
- `numberStyle` (page 741)
  Returns the number-formatter style of the receiver.
- `setGeneratesDecimalNumbers:` (page 749)
  Controls whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.
- `generatesDecimalNumbers` (page 731)
  Returns a Boolean value that indicates whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

Converting Between Numbers and Strings

- `getObjectValue:forString:range:error:` (page 732)
  Returns by reference a cell-content object after creating it from a range of characters in a given string.
- `numberFromString:` (page 741)
  Returns an `NSNumber` object created by parsing a given string.
- `stringFromNumber:` (page 767)
  Returns a string containing the formatted value of the provided number object.

Managing Localization of Numbers

- `setLocale:` (page 751)
  Sets the locale of the receiver.
- `locale` (page 734)
  Returns the locale of the receiver.

Configuring Rounding Behavior

- `setRoundingIncrement:` (page 762)
  Sets the rounding increment used by the receiver.
- `roundingIncrement` (page 744)
  Returns the rounding increment used by the receiver.
- `setRoundingMode:` (page 762)
  Sets the rounding mode used by the receiver.
- `roundingMode` (page 745)
  Returns the rounding mode used by the receiver.

Configuring Numeric Formats

- `formatWidth` (page 731)
  Returns the format width of the receiver.
- `setNegativeFormat:` (page 756)
  Sets the format the receiver uses to display negative values.
- `negativeFormat` (page 739)
  Returns the format used by the receiver to display negative numbers.
- `setPositiveFormat:` (page 761)
  Sets the format the receiver uses to display positive values.
- `positiveFormat` (page 743)
  Returns the format used by the receiver to display positive numbers.
- `setFormatWidth:` (page 749)
  Sets the format width used by the receiver.
- `setMultiplier:` (page 755)
  Sets the multiplier of the receiver.

- multiplier (page 738)
  Returns the multiplier used by the receiver as an NSNumber object.

Configuring Numeric Symbols

- percentSymbol (page 742)
  Returns the string that the receiver uses to represent the percent symbol.
- setPercentSymbol: (page 760)
  Sets the string used by the receiver to represent the percent symbol.
- perMillSymbol (page 742)
  Returns the string that the receiver uses for the per-thousands symbol.
- setPerMillSymbol: (page 760)
  Sets the string used by the receiver to represent the per-mill (per-thousand) symbol.
- minusSign (page 738)
  Returns the string the receiver uses to represent the minus sign.
- setMinusSign: (page 755)
  Sets the string used by the receiver for the minus sign.
- plusSign (page 743)
  Returns the string the receiver uses for the plus sign.
- setPlusSign: (page 760)
  Sets the string used by the receiver to represent the plus sign.
- exponentSymbol (page 730)
  Returns the string the receiver uses as an exponent symbol.
- setExponentSymbol: (page 748)
  Sets the string used by the receiver to represent the exponent symbol.
- zeroSymbol (page 771)
  Returns the string the receiver uses as the symbol to show the value zero.
- setZeroSymbol: (page 767)
  Sets the string the receiver uses as the symbol to show the value zero.
- nilSymbol (page 740)
  Returns the string the receiver uses to represent a nil value.
- setNilSymbol: (page 757)
  Sets the string the receiver uses to represent nil values.
- notANumberSymbol (page 740)
  Returns the symbol the receiver uses to represent NaN (“not a number”) when it converts values.
- setNotANumberSymbol: (page 758)
  Sets the string the receiver uses to represent NaN (“not a number”).
- negativeInfinitySymbol (page 739)
  Returns the symbol the receiver uses to represent negative infinity.
- setNegativeInfinitySymbol: (page 756)
  Sets the string used by the receiver for the negative infinity symbol.
- positiveInfinitySymbol (page 743)
  Returns the string the receiver uses for the positive infinity symbol.

- `setPositiveInfinitySymbol:` (page 761)
  Sets the string used by the receiver for the positive infinity symbol.

Configuring the Format of Currency

- `setCurrencySymbol:` (page 747)
  Sets the string used by the receiver as a local currency symbol.
- `currencySymbol` (page 729)
  Returns the receiver's local currency symbol.
- `setCurrencyCode:` (page 746)
  Sets the receiver's currency code.
- `currencyCode` (page 728)
  Returns the receiver's currency code as a string.
- `setInternationalCurrencySymbol:` (page 750)
  Sets the string used by the receiver for the international currency symbol.
- `internationalCurrencySymbol` (page 733)
  Returns the international currency symbol used by the receiver.
- `setCurrencyGroupingSeparator:` (page 747)
  Sets the currency grouping separator for the receiver.
- `currencyGroupingSeparator` (page 729)
  Returns the currency grouping separator for the receiver.

Configuring Numeric Prefixes and Suffixes

- `setPositivePrefix:` (page 761)
  Sets the string the receiver uses as the prefix for positive values.
- `positivePrefix` (page 744)
  Returns the string the receiver uses as the prefix for positive values.
- `setPositiveSuffix:` (page 762)
  Sets the string the receiver uses as the suffix for positive values.
- `positiveSuffix` (page 744)
  Returns the string the receiver uses as the suffix for positive values.
- `setNegativePrefix:` (page 757)
  Sets the string the receiver uses as a prefix for negative values.
- `negativePrefix` (page 739)
  Returns the string the receiver inserts as a prefix to negative values.
- `setNegativeSuffix:` (page 757)
  Sets the string the receiver uses as a suffix for negative values.
- `negativeSuffix` (page 740)
  Returns the string the receiver adds as a suffix to negative values.

Configuring the Display of Numeric Values

- `setTextAttributesForNegativeValues:` (page 764)
  Sets the text attributes to be used in displaying negative values .
- `textAttributesForNegativeValues` (page 768)
  Returns a dictionary containing the text attributes that have been set for negative values.
- `setTextAttributesForPositiveValues:` (page 765)
  Sets the text attributes to be used in displaying positive values.
- `textAttributesForPositiveValues` (page 770)
  Returns a dictionary containing the text attributes that have been set for positive values.
- `setTextAttributesForZero:` (page 766)
  Sets the text attributes used to display a zero value.
- `textAttributesForZero` (page 770)
  Returns a dictionary containing the text attributes used to display a value of zero.
- `setTextAttributesForNil:` (page 764)
  Sets the text attributes used to display the `nil` symbol.
- `textAttributesForNil` (page 769)
  Returns a dictionary containing the text attributes used to display the `nil` symbol.
- `setTextAttributesForNotANumber:` (page 765)
  Sets the text attributes used to display the NaN ("not a number") string.
- `textAttributesForNotANumber` (page 769)
  Returns a dictionary containing the text attributes used to display the NaN ("not a number") symbol.
- `setTextAttributesForPositiveInfinity:` (page 765)
  Sets the text attributes used to display the positive infinity symbol.
- `textAttributesForPositiveInfinity` (page 769)
  Returns a dictionary containing the text attributes used to display the positive infinity symbol.
- `setTextAttributesForNegativeInfinity:` (page 763)
  Sets the text attributes used to display the negative infinity symbol.
- `textAttributesForNegativeInfinity` (page 768)
  Returns a dictionary containing the text attributes used to display the negative infinity string.

Configuring Separators and Grouping Size

- `setGroupingSeparator:` (page 750)
  Specifies the string used by the receiver for a grouping separator.
- `groupingSeparator` (page 732)
  Returns a string containing the receiver's grouping separator.
- `setUsesGroupingSeparator:` (page 766)
  Controls whether the receiver displays the grouping separator.
- `usesGroupingSeparator` (page 770)
  Returns a Boolean value that indicates whether the receiver uses the grouping separator.
- `setDecimalSeparator:` (page 748)
  Sets the character the receiver uses as a decimal separator.

- `decimalSeparator` (page 730)
  Returns a string containing the character the receiver uses to represent decimal separators.
- `setAlwaysShowsDecimalSeparator:` (page 746)
  Controls whether the receiver always shows the decimal separator, even for integer numbers.
- `alwaysShowsDecimalSeparator` (page 728)
  Returns a Boolean value that indicates whether the receiver always shows a decimal separator, even if the number is an integer.
- `setCurrencyDecimalSeparator:` (page 747)
  Sets the string used by the receiver as a decimal separator.
- `currencyDecimalSeparator` (page 729)
  Returns the receiver's currency decimal separator as a string.
- `setGroupingSize:` (page 750)
  Sets the grouping size of the receiver.
- `groupingSize` (page 733)
  Returns the receiver's primary grouping size.
- `setSecondaryGroupingSize:` (page 763)
  Sets the secondary grouping size of the receiver.
- `secondaryGroupingSize` (page 745)
  Returns the size of secondary groupings for the receiver.

Managing the Padding of Numbers

- `setPaddingCharacter:` (page 758)
  Sets the string that the receiver uses to pad numbers in the formatted string representation.
- `paddingCharacter` (page 741)
  Returns a string containing the padding character for the receiver.
- `setPaddingPosition:` (page 759)
  Sets the padding position used by the receiver.
- `paddingPosition` (page 742)
  Returns the padding position of the receiver.

Managing Input Attributes

- `setAllowsFloats:` (page 745)
  Sets whether the receiver allows as input floating-point values (that is, values that include the period character [.]).
- `allowsFloats` (page 728)
  Returns a Boolean value that indicates whether the receiver allows floating-point values as input.
- `setMinimum:` (page 753)
  Sets the lowest number the receiver allows as input.
- `minimum` (page 736)
  Returns the lowest number allowed as input by the receiver.

- `setMaximum:` (page 752)
  Sets the highest number the receiver allows as input.
- `maximum` (page 735)
  Returns the highest number allowed as input by the receiver.
- `setMinimumIntegerDigits:` (page 754)
  Sets the minimum number of integer digits allowed as input by the receiver.
- `minimumIntegerDigits` (page 737)
  Returns the minimum number of integer digits allowed as input by the receiver.
- `setMinimumFractionDigits:` (page 754)
  Sets the minimum number of digits after the decimal separator allowed as input by the receiver.
- `minimumFractionDigits` (page 737)
  Returns the minimum number of digits after the decimal separator allowed as input by the receiver.
- `setMaximumIntegerDigits:` (page 753)
  Sets the maximum number of integer digits allowed as input by the receiver.
- `maximumIntegerDigits` (page 736)
  Returns the maximum number of integer digits allowed as input by the receiver.
- `setMaximumFractionDigits:` (page 752)
  Sets the maximum number of digits after the decimal separator allowed as input by the receiver.
- `maximumFractionDigits` (page 735)
  Returns the maximum number of digits after the decimal separator allowed as input by the receiver.

Configuring Significant Digits

- `setUsesSignificantDigits:` (page 767)
  Sets whether the receiver uses significant digits.
- `usesSignificantDigits` (page 771)
  Returns a Boolean value that indicates whether the receiver uses significant digits.
- `setMinimumSignificantDigits:` (page 755)
  Sets the minimum number of significant digits for the receiver.
- `minimumSignificantDigits` (page 737)
  Returns the minimum number of significant digits for the receiver.
- `setMaximumSignificantDigits:` (page 753)
  Sets the maximum number of significant digits for the receiver.
- `maximumSignificantDigits` (page 736)
  Returns the maximum number of significant digits for the receiver.

Managing Leniency Behavior

- `setLenient:` (page 751)
  Sets whether the receiver is will use heuristics to guess at the date which is intended by a string.

- isLenient (page 734)
  Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the date which is intended by a string.

Managing the Validation of Partial Numeric Strings

- setPartialStringValidationEnabled: (page 759)
  Sets whether partial string validation is enabled for the receiver.
- isPartialStringValidationEnabled (page 734)
  Returns a Boolean value that indicates whether partial string validation is enabled.

Class Methods

defaultFormatterBehavior

Returns an NSNumberFormatterBehavior constant that indicates default formatter behavior for new instances of NSNumberFormatter.

```
+ (NSNumberFormatterBehavior)defaultFormatterBehavior
```

Return Value

An NSNumberFormatterBehavior constant that indicates default formatter behavior for new instances of NSNumberFormatter.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ setDefaultFormatterBehavior: (page 727)

Declared In

NSNumberFormatter.h

setDefaultFormatterBehavior:

Sets the default formatter behavior for new instances of NSNumberFormatter .

```
+ (void)setDefaultFormatterBehavior:(NSNumberFormatterBehavior)behavior
```

Parameters

behavior
  An NSNumberFormatterBehavior constant that indicates the revision of the class providing the default behavior.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ defaultFormatterBehavior (page 727)

Declared In
NSNumberFormatter.h

Instance Methods

allowsFloats

Returns a Boolean value that indicates whether the receiver allows floating-point values as input.

```
- (BOOL)allowsFloats
```

Return Value
YES if the receiver allows as input floating-point values (that is, values that include the period character [.]), otherwise NO.

Discussion
When this method returns NO, only integer values can be provided as input. The default is YES.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setAllowsFloats: (page 745)

Declared In
NSNumberFormatter.h

alwaysShowsDecimalSeparator

Returns a Boolean value that indicates whether the receiver always shows a decimal separator, even if the number is an integer.

```
- (BOOL)alwaysShowsDecimalSeparator
```

Return Value
YES if the receiver always shows a decimal separator, even if the number is an integer, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setAlwaysShowsDecimalSeparator: (page 746)

Declared In
NSNumberFormatter.h

currencyCode

Returns the receiver's currency code as a string.

```
- (NSString *)currencyCode
```

Return Value
The receiver's currency code as a string.

Discussion
A currency code is a three-letter code that is, in most cases, composed of a country's two-character Internet country code plus an extra character to denote the currency unit. For example, the currency code for the Australian dollar is "AUD". Currency codes are based on the ISO 4217 standard.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setCurrencyCode:` (page 746)

Declared In
`NSNumberFormatter.h`

currencyDecimalSeparator

Returns the receiver's currency decimal separator as a string.

```
- (NSString *)currencyDecimalSeparator
```

Return Value
The receiver's currency decimal separator as a string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `currencyDecimalSeparator` (page 729)

Declared In
`NSNumberFormatter.h`

currencyGroupingSeparator

Returns the currency grouping separator for the receiver.

```
- (NSString *)currencyGroupingSeparator
```

Return Value
The currency grouping separator for the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSNumberFormatter.h`

currencySymbol

Returns the receiver's local currency symbol.

```
- (NSString *)currencySymbol
```

Discussion
A country typically has a local currency symbol and an international currency symbol. The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `internationalCurrencySymbol` (page 733)
- `setCurrencySymbol:` (page 747)

Declared In
`NSNumberFormatter.h`

decimalSeparator

Returns a string containing the character the receiver uses to represent decimal separators.

```
- (NSString *)decimalSeparator
```

Return Value
A string containing the character the receiver uses to represent decimal separators.

Discussion
The return value doesn't indicate whether decimal separators are enabled.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDecimalSeparator:` (page 748)

Declared In
`NSNumberFormatter.h`

exponentSymbol

Returns the string the receiver uses as an exponent symbol.

```
- (NSString *)exponentSymbol
```

Return Value
The string the receiver uses as an exponent symbol.

Discussion
The exponent symbol is the "E" or "e" in the scientific notation of numbers, as in 1.0e+56.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setExponentSymbol: (page 748)

Declared In
NSNumberFormatter.h

formatterBehavior

Returns an NSNumberFormatterBehavior constant that indicates the formatter behavior of the receiver.

```
- (NSNumberFormatterBehavior)formatterBehavior
```

Return Value
An NSNumberFormatterBehavior constant that indicates the formatter behavior of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setFormatterBehavior: (page 749)

Declared In
NSNumberFormatter.h

formatWidth

Returns the format width of the receiver.

```
- (NSUInteger)formatWidth
```

Discussion
The format width is the number of characters of a formatted number within a string that is either left justified or right justified based on the value returned from paddingPosition (page 742).

Availability
Available in iPhone OS 2.0 and later.

See Also
- setFormatWidth: (page 749)

Declared In
NSNumberFormatter.h

generatesDecimalNumbers

Returns a Boolean value that indicates whether the receiver creates instances of NSDecimalNumber when it converts strings to number objects.

```
- (BOOL)generatesDecimalNumbers
```

Return Value
YES if the receiver creates instances of NSDecimalNumber when it converts strings to number objects, NO if it creates instance of NSNumber.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setGeneratesDecimalNumbers: (page 749)

Declared In
NSNumberFormatter.h

getObjectValue:forString:range:error:
Returns by reference a cell-content object after creating it from a range of characters in a given string.

```
- (BOOL)getObjectValue:(out id *)anObject forString:(NSString *)aString range:(inout
    NSRange *)rangep error:(out NSError **)error
```

Parameters
anObject

On return, contains an instance of NSDecimalNumber or NSNumber based on the current value of generatesDecimalNumbers (page 731). The default is to return NSDecimalNumber instances

aString

A string object with the range of characters specified in *rangep* that is used to create *anObject*.

rangep

A range of characters in *aString*. On return, contains the actual range of characters used to create the object.

error

If an error occurs, upon return contains an NSError object that explains the reason why the conversion failed. If you pass in nil for *error* you are indicating that you are not interested in error information.

Return Value
YES if the conversion from string to cell-content object was successful, otherwise NO.

Discussion
If there is an error, the delegate (if any) of the control object managing the cell can then respond to the failure in the NSController delegation method control:didFailToFormatString:errorDescription:.

Availability
Available in iPhone OS 2.0 and later.

See Also
- numberFromString: (page 741)
- stringFromNumber: (page 767)

Declared In
NSNumberFormatter.h

groupingSeparator
Returns a string containing the receiver's grouping separator.

```
- (NSString *)groupingSeparator
```

Return Value
A string containing the receiver's grouping separator.

Discussion
For example, the grouping separator used in the United States is the comma ("10,000") whereas in France it is the period ("10.000").

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setGroupingSeparator:` (page 750)

Declared In
`NSNumberFormatter.h`

groupingSize

Returns the receiver's primary grouping size.

```
- (NSUInteger)groupingSize
```

Return Value
The receiver's primary grouping size.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setGroupingSize:` (page 750)

Declared In
`NSNumberFormatter.h`

internationalCurrencySymbol

Returns the international currency symbol used by the receiver.

```
- (NSString *)internationalCurrencySymbol
```

Discussion
A country typically has a local currency symbol and an international currency symbol. The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The international currency symbol is often represented by a Unicode code point.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `currencySymbol` (page 729)
- `setInternationalCurrencySymbol:` (page 750)

Declared In
NSNumberFormatter.h

isLenient

Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the date which is intended by a string.

```
- (BOOL)isLenient
```

Return Value
YES if the receiver uses heuristics to guess at the date which is intended by the string, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setLenient: (page 751)

Declared In
NSNumberFormatter.h

isPartialStringValidationEnabled

Returns a Boolean value that indicates whether partial string validation is enabled.

```
- (BOOL)isPartialStringValidationEnabled
```

Return Value
YES if partial string validation is enabled, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPartialStringValidationEnabled: (page 759)

Declared In
NSNumberFormatter.h

locale

Returns the locale of the receiver.

```
- (NSLocale *)locale
```

Return Value
The locale of the receiver.

Discussion
A number formatter's locale specifies default localization attributes, such as ISO country and language codes, currency code, calendar, system of measurement, and decimal separator.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setLocale:` (page 751)

Declared In
`NSNumberFormatter.h`

maximum

Returns the highest number allowed as input by the receiver.

```
- (NSNumber *)maximum
```

Return Value
The highest number allowed as input by the receiver or `nil`, meaning no limit.

Discussion
For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method returns an `NSDecimalNumber` object.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMaximum:` (page 752)
- `setDefaultFormatterBehavior:` (page 727)
- `formatterBehavior` (page 731)
- `setFormatterBehavior:` (page 749)

Declared In
`NSNumberFormatter.h`

maximumFractionDigits

Returns the maximum number of digits after the decimal separator allowed as input by the receiver.

```
- (NSUInteger)maximumFractionDigits
```

Return Value
The maximum number of digits after the decimal separator allowed as input by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMaximumFractionDigits:` (page 752)

Declared In
`NSNumberFormatter.h`

maximumIntegerDigits

Returns the maximum number of integer digits allowed as input by the receiver.

```
- (NSUInteger)maximumIntegerDigits
```

Return Value
The maximum number of integer digits allowed as input by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMaximumIntegerDigits:` (page 753)

Declared In
`NSNumberFormatter.h`

maximumSignificantDigits

Returns the maximum number of significant digits for the receiver.

```
- (NSUInteger)maximumSignificantDigits
```

Return Value
The maximum number of significant digits for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMaximumSignificantDigits:` (page 753)
- `minimumSignificantDigits` (page 737)
- `usesSignificantDigits` (page 771)

Declared In
`NSNumberFormatter.h`

minimum

Returns the lowest number allowed as input by the receiver.

```
- (NSNumber *)minimum
```

Return Value
The lowest number allowed as input by the receiver or `nil`, meaning no limit.

Discussion
For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method returns an `NSDecimalNumber` object.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- setMinimum:` (page 753)
- `+ setDefaultFormatterBehavior:` (page 727)
- `- formatterBehavior` (page 731)
- `- setFormatterBehavior:` (page 749)

Declared In
`NSNumberFormatter.h`

minimumFractionDigits

Returns the minimum number of digits after the decimal separator allowed as input by the receiver.

```
- (NSUInteger)minimumFractionDigits
```

Return Value
The minimum number of digits after the decimal separator allowed as input by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- setMinimumFractionDigits:` (page 754)

Declared In
`NSNumberFormatter.h`

minimumIntegerDigits

Returns the minimum number of integer digits allowed as input by the receiver.

```
- (NSUInteger)minimumIntegerDigits
```

Return Value
The minimum number of integer digits allowed as input by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- setMinimumIntegerDigits:` (page 754)

Declared In
`NSNumberFormatter.h`

minimumSignificantDigits

Returns the minimum number of significant digits for the receiver.

```
- (NSUInteger)minimumSignificantDigits
```

Return Value
The minimum number of significant digits for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMinimumSignificantDigits:` (page 755)
- `maximumSignificantDigits` (page 736)
- `usesSignificantDigits` (page 771)

Declared In
`NSNumberFormatter.h`

minusSign
Returns the string the receiver uses to represent the minus sign.

```
- (NSString *)minusSign
```

Return Value
The string that represents the receiver's minus sign.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMinusSign:` (page 755)

Declared In
`NSNumberFormatter.h`

multiplier
Returns the multiplier used by the receiver as an `NSNumber` object.

```
- (NSNumber *)multiplier
```

Discussion
A multiplier is a factor used in conversions between numbers and strings (that is, numbers as stored and numbers as displayed). When the input value is a string, the multiplier is used to divide, and when the input value is a number, the multiplier is used to multiply. These operations allow the formatted values to be different from the values that a program manipulates internally.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setMultiplier:` (page 755)

Declared In
`NSNumberFormatter.h`

negativeFormat

Returns the format used by the receiver to display negative numbers.

```
- (NSString *)negativeFormat
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- setNegativeFormat: (page 756)

Declared In
NSNumberFormatter.h

negativeInfinitySymbol

Returns the symbol the receiver uses to represent negative infinity.

```
- (NSString *)negativeInfinitySymbol
```

Return Value
The symbol the receiver uses to represent negative infinity.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setNegativeInfinitySymbol: (page 756)

Declared In
NSNumberFormatter.h

negativePrefix

Returns the string the receiver inserts as a prefix to negative values.

```
- (NSString *)negativePrefix
```

Return Value
The string the receiver inserts as a prefix to negative values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- negativeSuffix (page 740)
- setNegativePrefix: (page 757)

Declared In
NSNumberFormatter.h

negativeSuffix

Returns the string the receiver adds as a suffix to negative values.

```
- (NSString *)negativeSuffix
```

Return Value
The string the receiver adds as a suffix to negative values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `negativePrefix` (page 739)
- `setNegativeSuffix:` (page 757)

Declared In
`NSNumberFormatter.h`

nilSymbol

Returns the string the receiver uses to represent a `nil` value.

```
- (NSString *)nilSymbol
```

Return Value
The string the receiver uses to represent a `nil` value.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setNilSymbol:` (page 757)

Declared In
`NSNumberFormatter.h`

notANumberSymbol

Returns the symbol the receiver uses to represent NaN (“not a number”) when it converts values.

```
- (NSString *)notANumberSymbol
```

Return Value
The symbol the receiver uses to represent NaN (“not a number”) when it converts values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setNotANumberSymbol:` (page 758)

Declared In
`NSNumberFormatter.h`

numberFromString:

Returns an `NSNumber` object created by parsing a given string.

```
- (NSNumber *)numberFromString:(NSString *)string
```

Parameters

string

An `NSString` object that is parsed to generate the returned number object.

Return Value

An `NSNumber` object created by parsing *string* using the receiver's format.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `stringFromNumber:` (page 767)

Declared In

`NSNumberFormatter.h`

numberStyle

Returns the number-formatter style of the receiver.

```
- (NSNumberFormatterStyle)numberStyle
```

Return Value

An `NSNumberFormatterStyle` constant that indicates the number-formatter style of the receiver.

Discussion

Styles are essentially predetermined sets of values for certain properties. Examples of number-formatter styles are those used for decimal values, percentage values, and currency.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setNumberStyle:` (page 758)

Declared In

`NSNumberFormatter.h`

paddingCharacter

Returns a string containing the padding character for the receiver.

```
- (NSString *)paddingCharacter
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setPaddingCharacter:` (page 758)

Declared In
NSNumberFormatter.h

paddingPosition

Returns the padding position of the receiver.

```
- (NSNumberFormatterPadPosition)paddingPosition
```

Discussion
The returned constant indicates whether the padding is before or after the number's prefix or suffix.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPaddingPosition: (page 759)

Declared In
NSNumberFormatter.h

percentSymbol

Returns the string that the receiver uses to represent the percent symbol.

```
- (NSString *)percentSymbol
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPercentSymbol: (page 760)

Declared In
NSNumberFormatter.h

perMillSymbol

Returns the string that the receiver uses for the per-thousands symbol.

```
- (NSString *)perMillSymbol
```

Return Value
The string that the receiver uses for the per-thousands symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPerMillSymbol: (page 760)

Declared In
NSNumberFormatter.h

plusSign

Returns the string the receiver uses for the plus sign.

```
- (NSString *)plusSign
```

Return Value
The string the receiver uses for the plus sign.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPlusSign: (page 760)

Declared In
NSNumberFormatter.h

positiveFormat

Returns the format used by the receiver to display positive numbers.

```
- (NSString *)positiveFormat
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPositiveFormat: (page 761)

Declared In
NSNumberFormatter.h

positiveInfinitySymbol

Returns the string the receiver uses for the positive infinity symbol.

```
- (NSString *)positiveInfinitySymbol
```

Return Value
The string the receiver uses for the positive infinity symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPositiveInfinitySymbol: (page 761)

Declared In
NSNumberFormatter.h

positivePrefix

Returns the string the receiver uses as the prefix for positive values.

```
- (NSString *)positivePrefix
```

Return Value
The string the receiver uses as the prefix for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPositivePrefix: (page 761)

Declared In
NSNumberFormatter.h

positiveSuffix

Returns the string the receiver uses as the suffix for positive values.

```
- (NSString *)positiveSuffix
```

Return Value
The string the receiver uses as the suffix for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPositiveSuffix: (page 762)

Declared In
NSNumberFormatter.h

roundingIncrement

Returns the rounding increment used by the receiver.

```
- (NSNumber *)roundingIncrement
```

Return Value
The rounding increment used by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setRoundingIncrement: (page 762)

Declared In
NSNumberFormatter.h

roundingMode

Returns the rounding mode used by the receiver.

```
- (NSNumberFormatterRoundingMode)roundingMode
```

Return Value
The rounding mode used by the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setRoundingMode: (page 762)

Declared In
NSNumberFormatter.h

secondaryGroupingSize

Returns the size of secondary groupings for the receiver.

```
- (NSUInteger)secondaryGroupingSize
```

Return Value
The size of secondary groupings for the receiver.

Discussion
Some locales allow the specification of another grouping size for larger numbers. For example, some locales may represent a number such as 61, 242, 378.46 (as in the United States) as 6,12,42,378.46. In this case, the secondary grouping size (covering the groups of digits furthest from the decimal point) is 2.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setSecondaryGroupingSize: (page 763)

Declared In
NSNumberFormatter.h

setAllowsFloats:

Sets whether the receiver allows as input floating-point values (that is, values that include the period character [.]).

```
- (void)setAllowsFloats:(BOOL)flag
```

Parameters
flag

YES if the receiver allows floating-point values, NO otherwise.

Discussion
By default, floating point values are allowed as input.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allowsFloats` (page 728)

Declared In
`NSNumberFormatter.h`

setAlwaysShowsDecimalSeparator:

Controls whether the receiver always shows the decimal separator, even for integer numbers.

```
- (void)setAlwaysShowsDecimalSeparator:(BOOL)flag
```

Parameters

flag
`YES` if the receiver should always show the decimal separator, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `alwaysShowsDecimalSeparator` (page 728)

Declared In
`NSNumberFormatter.h`

setCurrencyCode:

Sets the receiver's currency code.

```
- (void)setCurrencyCode:(NSString *)string
```

Parameters

string
A string specifying the receiver's new currency code.

Discussion
A currency code is a three-letter code that is, in most cases, composed of a country's two-character Internet country code plus an extra character to denote the currency unit. For example, the currency code for the Australian dollar is "AUD". Currency codes are based on the ISO 4217 standard.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `currencyCode` (page 728)

Declared In
`NSNumberFormatter.h`

setCurrencyDecimalSeparator:

Sets the string used by the receiver as a decimal separator.

```
- (void)setCurrencyDecimalSeparator:(NSString *)string
```

Parameters

string

The string to use as the currency decimal separator.

Availability

Available in iPhone OS 2.0 and later.

See Also

- currencyDecimalSeparator (page 729)

Declared In

NSNumberFormatter.h

setCurrencyGroupingSeparator:

Sets the currency grouping separator for the receiver.

```
- (NSString *)setCurrencyGroupingSeparator:(NSString *)string
```

Parameters

string

The currency grouping separator for the receiver.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSNumberFormatter.h

setCurrencySymbol:

Sets the string used by the receiver as a local currency symbol.

```
- (void)setCurrencySymbol:(NSString *)string
```

Parameters

string

A string that represents a local currency symbol.

Discussion

The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability

Available in iPhone OS 2.0 and later.

See Also

- currencySymbol (page 729)

- setInternationalCurrencySymbol: (page 750)

Declared In
NSNumberFormatter.h

setDecimalSeparator:

Sets the character the receiver uses as a decimal separator.

- (void)setDecimalSeparator:(NSString *)newSeparator

Parameters

newSeparator

The string that specifies the decimal-separator character to use. If *newSeparator* contains multiple characters, only the first one is used.

Discussion
If you don't have decimal separators enabled through another means (such as setFormat:), using this method enables them.

Availability
Available in iPhone OS 2.0 and later.

See Also
- decimalSeparator (page 730)
- formatterBehavior (page 731)

Declared In
NSNumberFormatter.h

setExponentSymbol:

Sets the string used by the receiver to represent the exponent symbol.

- (void)setExponentSymbol:(NSString *)string

Parameters

string

A string that represents an exponent symbol.

Discussion
The exponent symbol is the "E" or "e" in the scientific notation of numbers, as in 1.0e+56.

Availability
Available in iPhone OS 2.0 and later.

See Also
- exponentSymbol (page 730)

Declared In
NSNumberFormatter.h

setFormatterBehavior:

Sets the formatter behavior of the receiver.

```
- (void)setFormatterBehavior:(NSNumberFormatterBehavior)behavior
```

Parameters

behavior

An `NSNumberFormatterBehavior` constant that indicates the revision of the `NSNumberFormatter` class providing the current behavior.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- formatterBehavior` (page 731)

Declared In

`NSNumberFormatter.h`

setFormatWidth:

Sets the format width used by the receiver.

```
- (void)setFormatWidth:(NSUInteger)number
```

Parameters

number

An integer that specifies the format width.

Discussion

The format width is the number of characters of a formatted number within a string that is either left justified or right justified based on the value returned from `paddingPosition` (page 742).

Availability

Available in iPhone OS 2.0 and later.

See Also

`- formatWidth` (page 731)

Declared In

`NSNumberFormatter.h`

setGeneratesDecimalNumbers:

Controls whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.

```
- (void)setGeneratesDecimalNumbers:(BOOL)flag
```

Parameters

flag

`YES` if the receiver should generate `NSDecimalNumber` instances, `NO` if it should generate `NSNumber` instances.

Discussion
The default is `YES`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- generatesDecimalNumbers` (page 731)

Declared In
`NSNumberFormatter.h`

setGroupingSeparator:

Specifies the string used by the receiver for a grouping separator.

```
- (void)setGroupingSeparator:(NSString *)string
```

Parameters

string
A string that specifies the grouping separator to use.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- groupingSeparator` (page 732)

Declared In
`NSNumberFormatter.h`

setGroupingSize:

Sets the grouping size of the receiver.

```
- (void)setGroupingSize:(NSUInteger)numDigits
```

Parameters

numDigits
An integer that specifies the grouping size.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- groupingSize` (page 733)

Declared In
`NSNumberFormatter.h`

setInternationalCurrencySymbol:

Sets the string used by the receiver for the international currency symbol.

```
- (void)setInternationalCurrencySymbol:(NSString *)string
```

Parameters

string

A string that represents an international currency symbol.

Discussion

The local symbol is used within the country, while the international currency symbol is used in international contexts to specify that country's currency unambiguously. The local currency symbol is often represented by a Unicode code point.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `internationalCurrencySymbol` (page 733)

Declared In

`NSNumberFormatter.h`

setLenient:

Sets whether the receiver is will use heuristics to guess at the date which is intended by a string.

```
- (void)setLenient:(BOOL)b
```

Parameters

b

`YES` if the receiver will use heuristics to guess at the date which is intended by the string, otherwise `NO`.

Discussion

If the formatter is set to be lenient, as with any guessing it may get the result date wrong (that is, a date other than that which was intended).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isLenient` (page 734)

Declared In

`NSNumberFormatter.h`

setLocale:

Sets the locale of the receiver.

```
- (void)setLocale:(NSLocale *)theLocale
```

Parameters

theLocale

An `NSLocale` object representing the new locale of the receiver.

Discussion
The locale determines the default values for many formatter attributes, such as ISO country and language codes, currency code, calendar, system of measurement, and decimal separator.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `locale` (page 734)

Declared In
`NSNumberFormatter.h`

setMaximum:

Sets the highest number the receiver allows as input.

```
- (void)setMaximum:(NSNumber *)aMaximum
```

Parameters

aMaximum
A number object that specifies a maximum input value.

Discussion
If *aMaximum* is `nil`, checking for the maximum value is disabled. For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method requires an `NSDecimalNumber` argument.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `maximum` (page 735)
+ `setDefaultFormatterBehavior:` (page 727)
- `formatterBehavior` (page 731)
- `setFormatterBehavior:` (page 749)

Declared In
`NSNumberFormatter.h`

setMaximumFractionDigits:

Sets the maximum number of digits after the decimal separator allowed as input by the receiver.

```
- (void)setMaximumFractionDigits:(NSUInteger)number
```

Parameters

number
The maximum number of digits after the decimal separator allowed as input.

Availability
Available in iPhone OS 2.0 and later.

See Also
- maximumFractionDigits (page 735)

Declared In
NSNumberFormatter.h

setMaximumIntegerDigits:

Sets the maximum number of integer digits allowed as input by the receiver.

```
- (void)setMaximumIntegerDigits:(NSUInteger)number
```

Parameters

number
The maximum number of integer digits allowed as input.

Availability
Available in iPhone OS 2.0 and later.

See Also
- minimumIntegerDigits (page 737)

Declared In
NSNumberFormatter.h

setMaximumSignificantDigits:

Sets the maximum number of significant digits for the receiver.

```
- (void)setMaximumSignificantDigits:(NSUInteger)number
```

Parameters

number
The maximum number of significant digits for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- maximumSignificantDigits (page 736)
- setMinimumSignificantDigits: (page 755)
- usesSignificantDigits (page 771)

Declared In
NSNumberFormatter.h

setMinimum:

Sets the lowest number the receiver allows as input.

```
- (void)setMinimum:(NSNumber *)aMinimum
```

Parameters

aMinimum

A number object that specifies a minimum input value.

Discussion

If *aMinimum* is `nil`, checking for the minimum value is disabled. For versions prior to Mac OS X v10.4 (and number-formatter behavior set to `NSNumberFormatterBehavior10_0`) this method requires an `NSDecimalNumber` argument.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `minimum` (page 736)
+ `setDefaultFormatterBehavior:` (page 727)
- `formatterBehavior` (page 731)
- `setFormatterBehavior:` (page 749)

Declared In

`NSNumberFormatter.h`

setMinimumFractionDigits:

Sets the minimum number of digits after the decimal separator allowed as input by the receiver.

```
- (void)setMinimumFractionDigits:(NSUInteger)number
```

Parameters

number

The minimum number of digits after the decimal separator allowed as input.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `minimumFractionDigits` (page 737)

Declared In

`NSNumberFormatter.h`

setMinimumIntegerDigits:

Sets the minimum number of integer digits allowed as input by the receiver.

```
- (void)setMinimumIntegerDigits:(NSUInteger)number
```

Parameters

number

The minimum number of integer digits allowed as input.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `minimumIntegerDigits` (page 737)

Declared In
`NSNumberFormatter.h`

setMinimumSignificantDigits:

Sets the minimum number of significant digits for the receiver.

```
- (void)setMinimumSignificantDigits:(NSUInteger)number
```

Parameters

number
The minimum number of significant digits for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `minimumSignificantDigits` (page 737)
- `setMaximumSignificantDigits:` (page 753)
- `usesSignificantDigits` (page 771)

Declared In
`NSNumberFormatter.h`

setMinusSign:

Sets the string used by the receiver for the minus sign.

```
- (void)setMinusSign:(NSString *)string
```

Parameters

string
A string that represents a minus sign.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `minusSign` (page 738)

Declared In
`NSNumberFormatter.h`

setMultiplier:

Sets the multiplier of the receiver.

```
- (void)setMultiplier:(NSNumber *)number
```

Parameters

number

A number object that represents a multiplier.

Discussion

A multiplier is a factor used in conversions between numbers and strings (that is, numbers as stored and numbers as displayed). When the input value is a string, the multiplier is used to divide, and when the input value is a number, the multiplier is used to multiply. These operations allow the formatted values to be different from the values that a program manipulates internally.

Availability

Available in iPhone OS 2.0 and later.

See Also

- multiplier (page 738)

Declared In

NSNumberFormatter.h

setNegativeFormat:

Sets the format the receiver uses to display negative values.

```
- (void)setNegativeFormat:(NSString *)aFormat
```

Parameters

aFormat

A string that specifies the format for negative values.

Availability

Available in iPhone OS 2.0 and later.

See Also

- negativeFormat (page 739)

Declared In

NSNumberFormatter.h

setNegativeInfinitySymbol:

Sets the string used by the receiver for the negative infinity symbol.

```
- (void)setNegativeInfinitySymbol:(NSString *)string
```

Parameters

string

A string that represents a negative infinity symbol.

Availability

Available in iPhone OS 2.0 and later.

See Also

- negativeInfinitySymbol (page 739)

Declared In
NSNumberFormatter.h

setNegativePrefix:

Sets the string the receiver uses as a prefix for negative values.

```
- (void)setNegativePrefix:(NSString *)string
```

Parameters

string
A string to use as the prefix for negative values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- negativePrefix (page 739)

Declared In
NSNumberFormatter.h

setNegativeSuffix:

Sets the string the receiver uses as a suffix for negative values.

```
- (void)setNegativeSuffix:(NSString *)string
```

Parameters

string
A string to use as the suffix for negative values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- negativeSuffix (page 740)

Declared In
NSNumberFormatter.h

setNilSymbol:

Sets the string the receiver uses to represent `nil` values.

```
- (void)setNilSymbol:(NSString *)string
```

Parameters

string
A string that represents a `nil` value.

Availability
Available in iPhone OS 2.0 and later.

See Also
- nilSymbol (page 740)

Declared In
NSNumberFormatter.h

setNotANumberSymbol:

Sets the string the receiver uses to represent NaN ("not a number").

```
- (void)setNotANumberSymbol:(NSString *)string
```

Parameters

string
A string that represents a NaN symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- notANumberSymbol (page 740)

Declared In
NSNumberFormatter.h

setNumberStyle:

Sets the number style used by the receiver.

```
- (void)setNumberStyle:(NSNumberFormatterStyle)style
```

Parameters

style
An NSNumberFormatterStyle constant that specifies a formatter style.

Discussion
Styles are essentially predetermined sets of values for certain properties. Examples of number-formatter styles are those used for decimal values, percentage values, and currency.

Availability
Available in iPhone OS 2.0 and later.

See Also
- numberStyle (page 741)

Declared In
NSNumberFormatter.h

setPaddingCharacter:

Sets the string that the receiver uses to pad numbers in the formatted string representation.

```
- (void)setPaddingCharacter:(NSString *)string
```

Parameters

string

A string containing a padding character (or characters).

Availability

Available in iPhone OS 2.0 and later.

See Also

- paddingCharacter (page 741)

Declared In

NSNumberFormatter.h

setPaddingPosition:

Sets the padding position used by the receiver.

```
- (void)setPaddingPosition:(NSNumberFormatterPadPosition)position
```

Parameters

position

An NSNumberFormatterPadPosition constant that indicates a padding position (before or after prefix or suffix).

Availability

Available in iPhone OS 2.0 and later.

See Also

- paddingPosition (page 742)

Declared In

NSNumberFormatter.h

setPartialStringValidationEnabled:

Sets whether partial string validation is enabled for the receiver.

```
- (void)setPartialStringValidationEnabled:(BOOL)b
```

Parameters

b

YES if partial string validation is enabled, otherwise NO.

Availability

Available in iPhone OS 2.0 and later.

See Also

- isPartialStringValidationEnabled (page 734)

Declared In

NSNumberFormatter.h

setPercentSymbol:

Sets the string used by the receiver to represent the percent symbol.

```
- (void)setPercentSymbol:(NSString *)string
```

Parameters

string
A string that represents a percent symbol.

Availability

Available in iPhone OS 2.0 and later.

See Also

- percentSymbol (page 742)

Declared In

NSNumberFormatter.h

setPerMillSymbol:

Sets the string used by the receiver to represent the per-mill (per-thousand) symbol.

```
- (void)setPerMillSymbol:(NSString *)string
```

Parameters

string
A string that represents a per-mill symbol.

Availability

Available in iPhone OS 2.0 and later.

See Also

- perMillSymbol (page 742)

Declared In

NSNumberFormatter.h

setPlusSign:

Sets the string used by the receiver to represent the plus sign.

```
- (void)setPlusSign:(NSString *)string
```

Parameters

string
A string that represents a plus sign.

Availability

Available in iPhone OS 2.0 and later.

See Also

- plusSign (page 743)

Declared In
NSNumberFormatter.h

setPositiveFormat:

Sets the format the receiver uses to display positive values.

```
- (void)setPositiveFormat:(NSString *)aFormat
```

Parameters

aFormat
A string that specifies the format for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- positiveFormat (page 743)

Declared In
NSNumberFormatter.h

setPositiveInfinitySymbol:

Sets the string used by the receiver for the positive infinity symbol.

```
- (void)setPositiveInfinitySymbol:(NSString *)string
```

Parameters

string
A string that represents a positive infinity symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- positiveInfinitySymbol (page 743)

Declared In
NSNumberFormatter.h

setPositivePrefix:

Sets the string the receiver uses as the prefix for positive values.

```
- (void)setPositivePrefix:(NSString *)string
```

Parameters

string
A string to use as the prefix for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- positivePrefix (page 744)

Declared In
NSNumberFormatter.h

setPositiveSuffix:

Sets the string the receiver uses as the suffix for positive values.

```
- (void)setPositiveSuffix:(NSString *)string
```

Parameters

string
A string to use as the suffix for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- positiveSuffix (page 744)

Declared In
NSNumberFormatter.h

setRoundingIncrement:

Sets the rounding increment used by the receiver.

```
- (void)setRoundingIncrement:(NSNumber *)number
```

Parameters

number
A number object specifying a rounding increment.

Availability
Available in iPhone OS 2.0 and later.

See Also
- roundingIncrement (page 744)

Declared In
NSNumberFormatter.h

setRoundingMode:

Sets the rounding mode used by the receiver.

```
- (void)setRoundingMode:(NSNumberFormatterRoundingMode)mode
```

Parameters

mode
An NSNumberFormatterRoundingMode constant that indicates a rounding mode.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `roundingMode` (page 745)

Declared In
`NSNumberFormatter.h`

setSecondaryGroupingSize:
Sets the secondary grouping size of the receiver.

```
- (void)setSecondaryGroupingSize:(NSUInteger)number
```

Parameters
number
An integer that specifies the size of secondary groupings.

Discussion
Some locales allow the specification of another grouping size for larger numbers. For example, some locales may represent a number such as 61, 242, 378.46 (as in the United States) as 6,12,42,378.46. In this case, the secondary grouping size (covering the groups of digits furthest from the decimal point) is 2.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `secondaryGroupingSize` (page 745)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNegativeInfinity:
Sets the text attributes used to display the negative infinity symbol.

```
- (void)setTextAttributesForNegativeInfinity:(NSDictionary *)newAttributes
```

Parameters
newAttributes
A dictionary containing text attributes for the display of the negative infinity symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `textAttributesForNegativeInfinity` (page 768)
- `setNegativeInfinitySymbol:` (page 756)

Declared In
`NSNumberFormatter.h`

setTextAttributesForNegativeValues:

Sets the text attributes to be used in displaying negative values .

```
- (void)setTextAttributesForNegativeValues:(NSDictionary *)newAttributes
```

Parameters

newAttributes

A dictionary containing properties for the display of negative values.

Discussion

For example, this code excerpt causes negative values to be displayed in red:

```
NSNumberFormatter *numberFormatter =
    [[[NSNumberFormatter alloc] init] autorelease];
NSMutableDictionary *newAttrs = [NSMutableDictionary dictionary];

[numberFormatter setFormat:@"$#,##0.00;($#,##0.00)"];
[newAttrs setObject:[NSColor redColor] forKey:@"NSColor"];
[numberFormatter setTextAttributesForNegativeValues:newAttrs];
[[textField cell] setFormatter:numberFormatter];
```

An even simpler way to cause negative values to be displayed in red is to include the constant `[Red]` in your format string, as shown in this example:

```
[numberFormatter setFormat:@"$#,##0.00;[Red]($#,##0.00)"];
```

When you set a value's text attributes to use color, the color appears only when the value's cell doesn't have input focus. When the cell has input focus, the value is displayed in standard black.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `textAttributesForNegativeValues` (page 768)

Declared In

`NSNumberFormatter.h`

setTextAttributesForNil:

Sets the text attributes used to display the `nil` symbol.

```
- (void)setTextAttributesForNil:(NSDictionary *)newAttributes
```

Parameters

newAttributes

A dictionary containing text attributes for the display of the `nil` symbol.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `textAttributesForNil` (page 769)
- `nilSymbol` (page 740)

Declared In
NSNumberFormatter.h

setTextAttributesForNotANumber:

Sets the text attributes used to display the NaN ("not a number") string.

```
- (void)setTextAttributesForNotANumber:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of the NaN symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setTextAttributesForNotANumber: (page 765)
- notANumberSymbol (page 740)

Declared In
NSNumberFormatter.h

setTextAttributesForPositiveInfinity:

Sets the text attributes used to display the positive infinity symbol.

```
- (void)setTextAttributesForPositiveInfinity:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of the positive infinity symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- positiveInfinitySymbol (page 743)
- textAttributesForPositiveInfinity (page 769)

Declared In
NSNumberFormatter.h

setTextAttributesForPositiveValues:

Sets the text attributes to be used in displaying positive values.

```
- (void)setTextAttributesForPositiveValues:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of positive values.

Discussion
See setTextAttributesForNegativeValues: (page 764) for an example of how a related method might be used.

Availability
Available in iPhone OS 2.0 and later.

See Also
- textAttributesForPositiveValues (page 770)

Declared In
NSNumberFormatter.h

setTextAttributesForZero:

Sets the text attributes used to display a zero value.

```
- (void)setTextAttributesForZero:(NSDictionary *)newAttributes
```

Parameters

newAttributes
A dictionary containing text attributes for the display of zero values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- textAttributesForZero (page 770)

Declared In
NSNumberFormatter.h

setUsesGroupingSeparator:

Controls whether the receiver displays the grouping separator.

```
- (void)setUsesGroupingSeparator:(BOOL)flag
```

Parameters

flag
YES if the receiver should display the grouping separator, NO otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
- usesGroupingSeparator (page 770)

Declared In
NSNumberFormatter.h

setUsesSignificantDigits:

Sets whether the receiver uses significant digits.

```
- (void)setUsesSignificantDigits:(BOOL)b
```

Parameters

b

`YES` if the receiver uses significant digits, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `usesSignificantDigits` (page 771)
- `setMaximumSignificantDigits:` (page 753)
- `setMinimumSignificantDigits:` (page 755)

Declared In

`NSNumberFormatter.h`

setZeroSymbol:

Sets the string the receiver uses as the symbol to show the value zero.

```
- (void)setZeroSymbol:(NSString *)string
```

Parameters

string

The string the receiver uses as the symbol to show the value zero.

Discussion

By default this is `0`; you might want to set it to, for example, " - ", similar to the way that a spreadsheet might when a column is defined as accounting.

Special Considerations

On Mac OS X v10.4, this method works correctly for 10_0-style number formatters but does not work correctly for 10_4-style number formatters. You can work around the problem by subclassing and overriding the methods that convert between strings and numbers to look for the zero cases first and provide different behavior, invoking `super` when not zero.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `zeroSymbol` (page 771)

Declared In

`NSNumberFormatter.h`

stringFromNumber:

Returns a string containing the formatted value of the provided number object.

```
- (NSString *)stringFromNumber:(NSNumber *)number
```

Parameters

number

An `NSNumber` object that is parsed to create the returned string object.

Return Value

A string containing the formatted value of *number* using the receiver's current settings.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `numberFromString:` (page 741)

Declared In

`NSNumberFormatter.h`

textAttributesForNegativeInfinity

Returns a dictionary containing the text attributes used to display the negative infinity string.

```
- (NSDictionary *)textAttributesForNegativeInfinity
```

Return Value

A dictionary containing the text attributes used to display the negative infinity string.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setTextAttributesForNegativeInfinity:` (page 763)

Declared In

`NSNumberFormatter.h`

textAttributesForNegativeValues

Returns a dictionary containing the text attributes that have been set for negative values.

```
- (NSDictionary *)textAttributesForNegativeValues
```

Return Value

A dictionary containing the text attributes that have been set for negative values.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setTextAttributesForNegativeValues:` (page 764)

Declared In

`NSNumberFormatter.h`

textAttributesForNil

Returns a dictionary containing the text attributes used to display the `nil` symbol.

```
- (NSDictionary *)textAttributesForNil
```

Return Value
A dictionary containing the text attributes used to display the `nil` symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTextAttributesForNil:` (page 764)

Declared In
`NSNumberFormatter.h`

textAttributesForNotANumber

Returns a dictionary containing the text attributes used to display the NaN ("not a number") symbol.

```
- (NSDictionary *)textAttributesForNotANumber
```

Return Value
A dictionary containing the text attributes used to display the NaN ("not a number") symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTextAttributesForNotANumber:` (page 765)
- `notANumberSymbol` (page 740)

Declared In
`NSNumberFormatter.h`

textAttributesForPositiveInfinity

Returns a dictionary containing the text attributes used to display the positive infinity symbol.

```
- (NSDictionary *)textAttributesForPositiveInfinity
```

Return Value
A dictionary containing the text attributes used to display the positive infinity symbol.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setTextAttributesForPositiveInfinity:` (page 765)
- `positiveInfinitySymbol` (page 743)

Declared In
NSNumberFormatter.h

textAttributesForPositiveValues

Returns a dictionary containing the text attributes that have been set for positive values.

```
- (NSDictionary *)textAttributesForPositiveValues
```

Return Value
A dictionary containing the text attributes that have been set for positive values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setTextAttributesForPositiveValues: (page 765)

Declared In
NSNumberFormatter.h

textAttributesForZero

Returns a dictionary containing the text attributes used to display a value of zero.

```
- (NSDictionary *)textAttributesForZero
```

Return Value
A dictionary containing the text attributes used to display a value of zero.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setTextAttributesForZero: (page 766)

Declared In
NSNumberFormatter.h

usesGroupingSeparator

Returns a Boolean value that indicates whether the receiver uses the grouping separator.

```
- (BOOL)usesGroupingSeparator
```

Return Value
YES if the receiver uses the grouping separator, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setUsesGroupingSeparator: (page 766)

Declared In
NSNumberFormatter.h

usesSignificantDigits

Returns a Boolean value that indicates whether the receiver uses significant digits.

```
- (BOOL)usesSignificantDigits
```

Return Value
YES if the receiver uses significant digits, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setUsesSignificantDigits: (page 767)
- maximumSignificantDigits (page 736)
- minimumSignificantDigits (page 737)

Declared In
NSNumberFormatter.h

zeroSymbol

Returns the string the receiver uses as the symbol to show the value zero.

```
- (NSString *)zeroSymbol
```

Return Value
The string the receiver uses as the symbol to show the value zero.

Discussion
For a discussion of how this is used, see setZeroSymbol: (page 767).

Availability
Available in iPhone OS 2.0 and later.

See Also
- setZeroSymbol: (page 767)

Declared In
NSNumberFormatter.h

Constants

NSNumberFormatterStyle

These constants specify predefined number format styles.

```
typedef enum {
    NSNumberFormatterNoStyle = kCFNumberFormatterNoStyle,
    NSNumberFormatterDecimalStyle = kCFNumberFormatterDecimalStyle,
    NSNumberFormatterCurrencyStyle = kCFNumberFormatterCurrencyStyle,
    NSNumberFormatterPercentStyle = kCFNumberFormatterPercentStyle,
    NSNumberFormatterScientificStyle = kCFNumberFormatterScientificStyle,
    NSNumberFormatterSpellOutStyle = kCFNumberFormatterSpellOutStyle
} NSNumberFormatterStyle;
```

Constants

`NSNumberFormatterNoStyle`

Specifies no style.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterDecimalStyle`

Specifies a decimal style format.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterCurrencyStyle`

Specifies a currency style format.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterPercentStyle`

Specifies a percent style format.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterScientificStyle`

Specifies a scientific style format.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterSpellOutStyle`

Specifies a spell-out format; for example, “23” becomes “twenty-three”.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

Discussion

These constants are used by the `numberStyle` (page 741) and `setNumberStyle:` (page 758) methods.

Declared In

`NSNumberFormatter.h`

NSNumberFormatterBehavior

These constants specify the behavior of a number formatter.

```
typedef enum {
    NSNumberFormatterBehaviorDefault = 0,
    NSNumberFormatterBehavior10_0 = 1000,
    NSNumberFormatterBehavior10_4 = 1040,
} NSNumberFormatterBehavior;
```

Constants

`NSNumberFormatterBehaviorDefault`

The number-formatter behavior set as the default for new instances. You can set the default formatter behavior with the class method `setDefaultFormatterBehavior:` (page 727).

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterBehavior10_0`

The number-formatter behavior as it existed prior to Mac OS X v10.4.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterBehavior10_4`

The number-formatter behavior since Mac OS X v10.4.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

Discussion

These constants are returned by the `defaultFormatterBehavior` (page 727) class method and the `formatterBehavior` (page 731) instance methods; you set them with the `setDefaultFormatterBehavior:` (page 727) class method and the `setFormatterBehavior:` (page 749) instance method.

Declared In

`NSNumberFormatter.h`

NSNumberFormatterPadPosition

These constants are used to specify how numbers should be padded.

```
typedef enum {
    NSNumberFormatterPadBeforePrefix = kCFNumberFormatterPadBeforePrefix,
    NSNumberFormatterPadAfterPrefix = kCFNumberFormatterPadAfterPrefix,
    NSNumberFormatterPadBeforeSuffix = kCFNumberFormatterPadBeforeSuffix,
    NSNumberFormatterPadAfterSuffix = kCFNumberFormatterPadAfterSuffix
} NSNumberFormatterPadPosition;
```

Constants

`NSNumberFormatterPadBeforePrefix`

Specifies that the padding should occur before the prefix.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterPadAfterPrefix`

Specifies that the padding should occur after the prefix.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterPadBeforeSuffix`
Specifies that the padding should occur before the suffix.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterPadAfterSuffix`
Specifies that the padding should occur after the suffix.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

Discussion
These constants are used by the `paddingPosition` (page 742) and `setPaddingPosition:` (page 759) methods.

Declared In
`NSNumberFormatter.h`

NSNumberFormatterRoundingMode
These constants are used to specify how numbers should be rounded.

```
typedef enum {
    NSNumberFormatterRoundCeiling = kCFNumberFormatterRoundCeiling,
    NSNumberFormatterRoundFloor = kCFNumberFormatterRoundFloor,
    NSNumberFormatterRoundDown = kCFNumberFormatterRoundDown,
    NSNumberFormatterRoundUp = kCFNumberFormatterRoundUp,
    NSNumberFormatterRoundHalfEven = kCFNumberFormatterRoundHalfEven,
    NSNumberFormatterRoundHalfDown = kCFNumberFormatterRoundHalfDown,
    NSNumberFormatterRoundHalfUp = kCFNumberFormatterRoundHalfUp
} NSNumberFormatterRoundingMode;
```

Constants
`NSNumberFormatterRoundCeiling`
Round up to next larger number with the proper number of digits after the decimal separator.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundFloor`
Round down to next smaller number with the proper number of digits after the decimal separator.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundDown`
Round down to next smaller number with the proper number of digits after the decimal separator.

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundHalfEven`
Round the last digit, when followed by a 5, toward an even digit (.25 -> .2, .35 -> .4)

Available in iPhone OS 2.0 and later.

Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundUp`
: Round up to next larger number with the proper number of digits after the decimal separator.

  Available in iPhone OS 2.0 and later.

  Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundHalfDown`
: Round down when a 5 follows putative last digit.

  Available in iPhone OS 2.0 and later.

  Declared in `NSNumberFormatter.h`

`NSNumberFormatterRoundHalfUp`
: Round up when a 5 follows putative last digit.

  Available in iPhone OS 2.0 and later.

  Declared in `NSNumberFormatter.h`

Declared In
`NSNumberFormatter.h`

These constants are used by the `roundingMode` (page 745)and `setRoundingMode:` (page 762) methods.

NSObject Class Reference

Inherits from:	none (NSObject is a root class)
Conforms to:	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSObject.h Foundation/NSArchiver.h Foundation/NSClassDescription.h Foundation/NSConnection.h Foundation/NSKeyedArchiver.h Foundation/NSObjectScripting.h Foundation/NSPortCoder.h Foundation/NSRunLoop.h Foundation/NSScriptClassDescription.h Foundation/NSThread.h
Companion guide:	Cocoa Fundamentals Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSObject` is the root class of most Objective-C class hierarchies. Through `NSObject`, objects inherit a basic interface to the runtime system and the ability to behave as Objective-C objects.

Selectors

NSObject has some special methods that take advantage of the Objective-C runtime system. For example, you can ask a class or instance if it responds to a message before sending it a message. You can also ask for a method implementation and invoke it using one of the perform... methods, or as a function. The advantage of obtaining a method's implementation and calling it as a function is that you can invoke the implementation multiple times within a loop, or similar C construct, without the overhead of Objective-C messaging.

These and other NSObject methods take a selector of type SEL as an argument. For efficiency, full ASCII names are not used to represent methods in compiled code. Instead the compiler uses a unique identifier to represent a method at runtime called a **selector**. A selector for a method name is obtained using the @selector() directive:

```
SEL method = @selector(isEqual:);
```

The instanceMethodForSelector: (page 790) class method and the methodForSelector: (page 805) instance method return a method implementation of type IMP. IMP is defined as a pointer to a function that returns an id and takes a variable number of arguments (in addition to the two "hidden" arguments—self and _cmd—that are passed to every method implementation):

```
typedef id (*IMP)(id, SEL, ...);
```

This definition serves as a prototype for the function pointer returned by these methods. It's sufficient for methods that return an object and take object arguments. However, if the selector takes different argument types or returns anything but an id, its function counterpart will be inadequately prototyped. Lacking a prototype, the compiler will promote floats to doubles and chars to ints, which the implementation won't expect. It will therefore behave differently (and erroneously) when performed as a method.

To remedy this situation, it's necessary to provide your own prototype. In the example below, the declaration of the test variable serves to prototype the implementation of the isEqual: method. test is defined as a pointer to a function that returns a BOOL and takes an id argument (in addition to the two "hidden" arguments). The value returned by methodForSelector: (page 805) is then similarly cast to be a pointer to this same function type:

```
BOOL (*test)(id, SEL, id);
test = (BOOL (*)(id, SEL, id))[target methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    ...
}
```

In some cases, it might be clearer to define a type (similar to IMP) that can be used both for declaring the variable and for casting the function pointer methodForSelector: (page 805) returns. The example below defines the EqualIMP type for just this purpose:

```
typedef BOOL (*EqualIMP)(id, SEL, id);
EqualIMP test;
test = (EqualIMP)[target methodForSelector:@selector(isEqual:)];

while ( !test(target, @selector(isEqual:), someObject) ) {
    ...
}
```

Either way, it's important to cast the return value of `methodForSelector:` (page 805) to the appropriate function type. It's not sufficient to simply call the function returned by `methodForSelector:` and cast the result of that call to the desired type. Doing so can result in errors.

See "How Messaging Works" in *The Objective-C 2.0 Programming Language* for more information.

Adopted Protocols

NSObject
- `autorelease` (page 1303)
- `class` (page 1304)
- `conformsToProtocol:` (page 1304)
- `description` (page 1305)
- `hash` (page 1305)
- `isEqual:` (page 1306)
- `isKindOfClass:` (page 1306)
- `isMemberOfClass:` (page 1307)
- `isProxy` (page 1308)
- `performSelector:` (page 1308)
- `performSelector:withObject:` (page 1309)
- `performSelector:withObject:withObject:` (page 1309)
- `release` (page 1310)
- `respondsToSelector:` (page 1311)
- `retain` (page 1312)
- `retainCount` (page 1312)
- `self` (page 1313)
- `superclass` (page 1313)
- `zone` (page 1314)

Tasks

Initializing a Class

`+ initialize` (page 788)
: Initializes the receiver before it's used (before it receives its first message).

`+ load` (page 792)
: Invoked whenever a class or category is added to the Objective-C runtime; implement this method to perform class-specific behavior upon loading.

Creating, Copying, and Deallocating Objects

+ `new` (page 793)
  Allocates a new instance of the receiving class, sends it an `init` (page 803) message, and returns the initialized object.

+ `alloc` (page 783)
  Returns a new instance of the receiving class.

+ `allocWithZone:` (page 783)
  Returns a new instance of the receiving class where memory for the new instance is allocated from a given zone.

- `init` (page 803)
  Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

- `copy` (page 798)
  Returns the object returned by `copyWithZone:` (page 1250), where the zone is `nil`.

+ `copyWithZone:` (page 787)
  Returns the receiver.

- `mutableCopy` (page 806)
  Returns the object returned by `mutableCopyWithZone:` (page 1300) where the zone is `nil`.

+ `mutableCopyWithZone:` (page 792)
  Returns the receiver.

- `dealloc` (page 799)
  Deallocates the memory occupied by the receiver.

- `finalize` (page 800)
  The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Identifying Classes

+ `class` (page 785)
  Returns the class object.

+ `superclass` (page 796)
  Returns the class object for the receiver's superclass.

+ `isSubclassOfClass:` (page 791)
  Returns a Boolean value that indicates whether the receiving class is a subclass of, or identical to, a given class.

Testing Class Functionality

+ `instancesRespondToSelector:` (page 791)
  Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

Testing Protocol Conformance

`+ conformsToProtocol:` (page 787)
Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Obtaining Information About Methods

`- methodForSelector:` (page 805)
Locates and returns the address of the receiver's implementation of a method so it can be called as a function.

`+ instanceMethodForSelector:` (page 790)
Locates and returns the address of the implementation of the instance method identified by a given selector.

`+ instanceMethodSignatureForSelector:` (page 790)
Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.

`- methodSignatureForSelector:` (page 805)
Returns an `NSMethodSignature` object that contains a description of the method identified by a given selector.

Describing Objects

`+ description` (page 788)
Returns a string that represents the contents of the receiving class.

Sending Messages

`- performSelector:withObject:afterDelay:` (page 808)
Invokes a method of the receiver on the current thread using the default mode after a delay.

`- performSelector:withObject:afterDelay:inModes:` (page 809)
Invokes a method of the receiver on the current thread using the specified modes after a delay.

`- performSelectorOnMainThread:withObject:waitUntilDone:` (page 811)
Invokes a method of the receiver on the main thread using the default mode.

`- performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812)
Invokes a method of the receiver on the main thread using the specified modes.

`- performSelector:onThread:withObject:waitUntilDone:` (page 806)
Invokes a method of the receiver on the specified thread using the default mode.

`- performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)
Invokes a method of the receiver on the specified thread using the specified modes.

`- performSelectorInBackground:withObject:` (page 810)
Invokes a method of the receiver on a new background thread.

`+ cancelPreviousPerformRequestsWithTarget:` (page 784)
Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method.

+ cancelPreviousPerformRequestsWithTarget:selector:object: (page 785)
Cancels perform requests previously registered with performSelector:withObject:afterDelay: (page 808).

Forwarding Messages

- forwardInvocation: (page 801)
Overridden by subclasses to forward messages to other objects.

Dynamically Resolving Methods

+ resolveClassMethod: (page 794)
Dynamically provides an implementation for a given selector for a class method.

+ resolveInstanceMethod: (page 794)
Dynamically provides an implementation for a given selector for an instance method.

Error Handling

- doesNotRecognizeSelector: (page 799)
Handles messages the receiver doesn't recognize.

Archiving

- awakeAfterUsingCoder: (page 797)
Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.

- classForCoder (page 797)
Overridden by subclasses to substitute a class other than its own during coding.

- classForKeyedArchiver (page 798)
Overridden by subclasses to substitute a new class for instances during keyed archiving.

+ classFallbacksForKeyedArchiver (page 786)
Overridden to return the names of classes that can be used to decode objects if their class is unavailable.

+ classForKeyedUnarchiver (page 786)
Overridden by subclasses to substitute a new class during keyed unarchiving.

- replacementObjectForCoder: (page 813)
Overridden by subclasses to substitute another object for itself during encoding.

- replacementObjectForKeyedArchiver: (page 814)
Overridden by subclasses to substitute another object for itself during keyed archiving.

+ setVersion: (page 795)
Sets the receiver's version number.

+ version (page 796)
Returns the version number assigned to the class.

Class Methods

alloc

Returns a new instance of the receiving class.

```
+ (id)alloc
```

Return Value

A new instance of the receiver.

Discussion

The `isa` instance variable of the new instance is initialized to a data structure that describes the class; memory for all other instance variables is set to `0`. The new instance is allocated from the default zone—use `allocWithZone:` (page 783) to specify a particular zone.

An `init...` method must be used to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass alloc] init];
```

Subclasses shouldn't override `alloc` to include initialization code. Instead, class-specific versions of `init...` methods should be implemented for that purpose. Class methods can also be implemented to combine allocation and initialization, similar to the `new` class method.

Special Considerations

If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of `1` and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 1310) or `autorelease` (page 1303).

Availability

Available in iPhone OS 2.0 and later.

See Also

`- init` (page 803)

Declared In

`NSObject.h`

allocWithZone:

Returns a new instance of the receiving class where memory for the new instance is allocated from a given zone.

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

Parameters

zone

The memory zone in which to create the new instance.

Return Value

A new instance of the receiver, where memory for the new instance is allocated from *zone*.

Discussion

The `isa` instance variable of the new instance is initialized to a data structure that describes the class; memory for its other instance variables is set to 0. If *zone* is `nil`, the new instance will be allocated from the default zone (as returned by `NSDefaultMallocZone`).

An `init...` method must be used to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass allocWithZone:someZone] init];
```

Subclasses shouldn't override `allocWithZone:` to include any initialization code. Instead, class-specific versions of `init...` methods should be implemented for that purpose.

When one object creates another, it's sometimes a good idea to make sure they're both allocated from the same region of memory. The `zone` (page 1314) method (declared in the `NSObject` protocol) can be used for this purpose; it returns the zone where the receiver is located. For example:

```
id myCompanion = [[TheClass allocWithZone:[self zone]] init];
```

Special Considerations

If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of `1` and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 1310) or `autorelease` (page 1303).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ alloc` (page 783)
`- init` (page 803)

Declared In

`NSObject.h`

cancelPreviousPerformRequestsWithTarget:

Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method.

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method.

Discussion

All perform requests having the same target *aTarget* are canceled. This method removes perform requests only in the current run loop, not all run loops.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRunLoop.h`

cancelPreviousPerformRequestsWithTarget:selector:object:

Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 808).

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget selector:(SEL)aSelector
    object:(id)anArgument
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method

aSelector

The selector for requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method.

See "Selectors" (page 778) for a description of the `SEL` type.

anArgument

The argument for requests previously registered with the `performSelector:withObject:afterDelay:` (page 808) instance method. Argument equality is determined using `isEqual:` (page 1306), so the value need not be the same object that was passed originally. Pass `nil` to match a request for `nil` that was originally passed as the argument.

Discussion

All perform requests are canceled that have the same target as *aTarget*, argument as *anArgument*, and selector as *aSelector*. This method removes perform requests only in the current run loop, not all run loops.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRunLoop.h`

class

Returns the class object.

```
+ (Class)class
```

Return Value

The class object.

Discussion

Refer to a class only by its name when it is the receiver of a message. In all other cases, the class object must be obtained through this or a similar method. For example, here `SomeClass` is passed as an argument to the `isKindOfClass:` (page 1306) method (declared in the `NSObject` protocol):

```
BOOL test = [self isKindOfClass:[SomeClass class]];
```

Availability

Available in iPhone OS 2.0 and later.

See Also
`class` (page 1304) (`NSObject` protocol)

Declared In
`NSObject.h`

classFallbacksForKeyedArchiver

Overridden to return the names of classes that can be used to decode objects if their class is unavailable.

```
+ (NSArray *)classFallbacksForKeyedArchiver
```

Return Value
An array of `NSString` objects that specify the names of classes in preferred order for unarchiving

Discussion
`NSKeyedArchiver` calls this method and stores the result inside the archive. If the actual class of an object doesn't exist at the time of unarchiving, `NSKeyedUnarchiver` goes through the stored list of classes and uses the first one that does exists as a substitute class for decoding the object. The default implementation of this method returns `nil`.

Developers who introduce a new class can use this method to provided some backwards compatibility in case the archive will be read on a system that does not have that class. Sometimes there may be another class which may work nearly as well as a substitute for the new class, and the archive keys and archived state for the new class can be carefully chosen (or compatibility written out) so that the object can be unarchived as the substitute class if necessary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

classForKeyedUnarchiver

Overridden by subclasses to substitute a new class during keyed unarchiving.

```
+ (Class)classForKeyedUnarchiver
```

Return Value
The class to substitute for the receiver during keyed unarchiving.

Discussion
During keyed unarchiving, instances of the receiver will be decoded as members of the returned class. This method overrides the results of the decoder's class and instance name to class encoding tables.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyedArchiver.h`

conformsToProtocol:

Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

```
+ (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Parameters

aProtocol
A protocol.

Return Value

`YES` if the receiver conforms to *aProtocol*, otherwise `NO`.

Discussion

A class is said to "conform to" a protocol if it adopts the protocol or inherits from another class that adopts it. Protocols are adopted by listing them within angle brackets after the interface declaration. For example, here `MyClass` adopts the (fictitious) `AffiliationRequests` and `Normalization` protocols:

```
@interface MyClass : NSObject <AffiliationRequests, Normalization>
```

A class also conforms to any protocols that are incorporated in the protocols it adopts or inherits. Protocols incorporate other protocols in the same way classes adopt them. For example, here the `AffiliationRequests` protocol incorporates the `Joining` protocol:

```
@protocol AffiliationRequests <Joining>
```

If a class adopts a protocol that incorporates another protocol, it must also implement all the methods in the incorporated protocol or inherit those methods from a class that adopts it.

This method determines conformance solely on the basis of the formal declarations in header files, as illustrated above. It doesn't check to see whether the methods declared in the protocol are actually implemented—that's the programmer's responsibility.

The protocol required as this method's argument can be specified using the `@protocol()` directive:

```
BOOL canJoin = [MyClass conformsToProtocol:@protocol(Joining)];
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ conformsToProtocol:` (page 787)

Declared In

`NSObject.h`

copyWithZone:

Returns the receiver.

```
+ (id)copyWithZone:(NSZone *)zone
```

Return Value

The receiver.

Discussion
This method exists so class objects can be used in situations where you need an object that conforms to the `NSCopying` protocol. For example, this method lets you use a class object as a key to an `NSDictionary` object. You should not override this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- copy` (page 798)

Declared In
`NSObject.h`

description

Returns a string that represents the contents of the receiving class.

```
+ (NSString *)description
```

Return Value
A string that represents the contents of the receiving class.

Discussion
The debugger's print-object command invokes this method to produce a textual description of an object.

`NSObject`'s implementation of this method simply prints the name of the class.

Availability
Available in iPhone OS 2.0 and later.

See Also
`description` (page 1305) (`NSObject` protocol)

Declared In
`NSObject.h`

initialize

Initializes the receiver before it's used (before it receives its first message).

```
+ (void)initialize
```

Discussion
The runtime sends `initialize` to each class in a program exactly one time just before the class, or any class that inherits from it, is sent its first message from within the program. (Thus the method may never be invoked if the class is not used.) The runtime sends the `initialize` message to classes in a thread-safe manner. Superclasses receive this message before their subclasses.

For example, if the first message your program sends is this:

```
[NSApplication new]
```

the runtime system sends these three `initialize` messages:

```
[NSObject initialize];
[NSResponder initialize];
[NSApplication initialize];
```

because `NSApplication` is a subclass of `NSResponder` and `NSResponder` is a subclass of `NSObject`. All the `initialize` messages precede the `new` (page 793) message.

If your program later begins to use the `NSText` class,

```
[NSText instancesRespondToSelector:someSelector]
```

the runtime system invokes these additional `initialize` messages:

```
[NSView initialize];
[NSText initialize];
```

because `NSText` inherits from `NSObject`, `NSResponder`, and `NSView`. The `instancesRespondToSelector:` (page 791) message is sent only after all these classes are initialized. Note that the `initialize` messages to `NSObject` and `NSResponder` aren't repeated.

You implement `initialize` to provide class-specific initialization as needed. Since the runtime sends appropriate initialize messages automatically, you should typically not send `initialize` to `super` in your implementation.

If a particular class does not implement `initialize`, the `initialize` method of its superclass is invoked twice, once for the superclass and once for the non-implementing subclass. If you want to make sure that your class performs class-specific initializations only once, implement `initialize` as in the following example:

```
@implementation MyClass
+ (void)initialize
{
    if ( self == [MyClass class] ) {
        /* put initialization code here */
    }
}
```

Loading a subclasses of MyClass that does not implement its own `initialize` method will cause MyClass's implementation to be invoked. The test clause (`if ( self == [MyClass class] )`) ensures that the initialization code has no effect if `initialize` is invoked when a subclass is loaded.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- init` (page 803)
`class` (page 1304) (`NSObject` protocol)

Declared In
`NSObject.h`

instanceMethodForSelector:

Locates and returns the address of the implementation of the instance method identified by a given selector.

```
+ (IMP)instanceMethodForSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies the method for which to return the implementation address. The selector must be non-NULL and valid for the receiver. If in doubt, use the `respondsToSelector:` (page 1311) method to check before passing the selector to `methodForSelector:`.

See "Selectors" (page 778) for a description of the `SEL` type.

Return Value

The address of the implementation of the *aSelector* instance method.

Discussion

An error is generated if instances of the receiver can't respond to *aSelector* messages.

Use this method to ask the class object for the implementation of instance methods only. To ask the class for the implementation of a class method, send the `methodForSelector:` (page 805) instance method to the class instead.

See "Selectors" (page 778) for a description of the `IMP` type, and how to invoke the returned method implementation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSObject.h`

instanceMethodSignatureForSelector:

Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.

```
+ (NSMethodSignature *)instanceMethodSignatureForSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies the method for which to return the implementation address.

See "Selectors" (page 778) for a description of the `SEL` type.

Return Value

An `NSMethodSignature` object that contains a description of the instance method identified by *aSelector*, or `nil` if the method can't be found.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- methodSignatureForSelector:` (page 805)

Declared In
NSObject.h

instancesRespondToSelector:

Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

```
+ (BOOL)instancesRespondToSelector:(SEL)aSelector
```

Parameters

aSelector
A selector. See "Selectors" (page 778) for a description of the SEL type.

Return Value
YES if instances of the receiver are capable of responding to *aSelector* messages, otherwise NO.

Discussion
If *aSelector* messages are forwarded to other objects, instances of the class are able to receive those messages without error even though this method returns NO.

To ask the class whether it, rather than its instances, can respond to a particular message, send to the class instead the NSObject protocol instance method respondsToSelector: (page 1311).

Availability
Available in iPhone OS 2.0 and later.

See Also
- forwardInvocation: (page 801)

Declared In
NSObject.h

isSubclassOfClass:

Returns a Boolean value that indicates whether the receiving class is a subclass of, or identical to, a given class.

```
+ (BOOL)isSubclassOfClass:(Class)aClass
```

Parameters

aClass
A class object.

Return Value
YES if the receiving class is a subclass of—or identical to—*aClass*, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

load

Invoked whenever a class or category is added to the Objective-C runtime; implement this method to perform class-specific behavior upon loading.

```
+ (void)load
```

Discussion

The `load` message is sent to classes and categories that are both dynamically loaded and statically linked, but only if the newly loaded class or category implements a method that can respond.

On Mac OS X v10.5, the order of initialization is as follows:

1. All initializers in any framework you link to.
2. All `+load` methods in your image.
3. All C++ static initializers and C/C++ `__attribute__(constructor)` functions in your image.
4. All initializers in frameworks that link to you.

In addition:

- A class's `+load` method is called after all of its superclasses' `+load` methods.
- A category `+load` method is called after the class's own `+load` method.

In a `+load` method, you can therefore safely message other unrelated classes from the same image, but any `+load` methods on those classes may not have run yet.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSObject.h`

mutableCopyWithZone:

Returns the receiver.

```
+ (id)mutableCopyWithZone:(NSZone *)zone
```

Parameters

zone

The memory zone in which to create the copy of the receiver.

Return Value

The receiver.

Discussion

This method exists so class objects can be used in situations where you need an object that conforms to the `NSMutableCopying` protocol. For example, this method lets you use a class object as a key to an `NSDictionary` object. You should not override this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

new
Allocates a new instance of the receiving class, sends it an init (page 803) message, and returns the initialized object.

```
+ (id)new
```

Return Value
A new instance of the receiver.

Discussion
This method is a combination of alloc (page 783) and init (page 803). Like alloc (page 783), it initializes the isa instance variable of the new object so it points to the class data structure. It then invokes the init (page 803) method to complete the initialization process.

Unlike alloc (page 783), new (page 793) is sometimes re-implemented in subclasses to invoke a class-specific initialization method. If the init... method includes arguments, they're typically reflected in a new... method as well. For example:

```
+ newMyClassWithTag:(int)tag data:(struct info *)data
{
    return [[self alloc] initWithTag:tag data:data];
}
```

However, there's little point in implementing a new... method if it's simply a shorthand for alloc (page 783) and init..., as shown above. Often new... methods will do more than just allocation and initialization. In some classes, they manage a set of instances, returning the one with the requested properties if it already exists, allocating and initializing a new instance only if necessary. For example:

```
+ newMyClassWithTag:(int)tag data:(struct info *)data
{
    MyClass *theInstance;

    if ( theInstance = findTheObjectWithTheTag(tag) )
        return theInstance;
    return [[self alloc] initWithTag:tag data:data];
}
```

Although it's appropriate to define new new... methods in this way, the alloc (page 783) and allocWithZone: (page 783) methods should never be augmented to include initialization code.

Special Considerations
If you are using managed memory (not garbage collection), this method retains the object before returning it. The returned object has a retain count of 1 and is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either release (page 1310) or autorelease (page 1303).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

resolveClassMethod:

Dynamically provides an implementation for a given selector for a class method.

```
+ (BOOL)resolveClassMethod:(SEL)name
```

Parameters

name

The name of a selector to resolve.

Return Value
YES if the method was found and added to the receiver, otherwise NO.

Discussion
This method allows you to dynamically provides an implementation for a given selector. See resolveInstanceMethod: (page 794) for further discussion.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ resolveInstanceMethod: (page 794)

Declared In
NSObject.h

resolveInstanceMethod:

Dynamically provides an implementation for a given selector for an instance method.

```
+ (BOOL)resolveInstanceMethod:(SEL)name
```

Parameters

name

The name of a selector to resolve.

Return Value
YES if the method was found and added to the receiver, otherwise NO.

Discussion
This method and resolveClassMethod: (page 794) allow you to dynamically provide an implementation for a given selector.

An Objective-C method is simply a C function that take at least two arguments—self and _cmd. Using the class_addMethod function, you can add a function to a class as a method. Given the following function:

```
void dynamicMethodIMP(id self, SEL _cmd)
{
    // implementation ....
}
```

you can use `resolveInstanceMethod:` to dynamically add it to a class as a method (called `resolveThisMethodDynamically`) like this:

```
+ (BOOL) resolveInstanceMethod:(SEL)aSEL
{
    if (aSEL == @selector(resolveThisMethodDynamically))
    {
          class_addMethod([self class], aSEL, (IMP) dynamicMethodIMP, "v@:");
          return YES;
    }
    return [super resolveInstanceMethod:aSel];
}
```

Special Considerations

This method is called before the Objective-C forwarding mechanism (see The Runtime System in *The Objective-C 2.0 Programming Language*) is invoked. If `respondsToSelector:` (page 1311) or `instancesRespondToSelector:` (page 791) is invoked, the dynamic method resolver is given the opportunity to provide an `IMP` for the given selector first.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ resolveClassMethod:` (page 794)

Declared In

`NSObject.h`

setVersion:

Sets the receiver's version number.

```
+ (void)setVersion:(NSInteger)aVersion
```

Parameters

aVersion

The version number for the receiver.

Discussion

The version number is helpful when instances of the class are to be archived and reused later. The default version is `0`.

Special Considerations

The version number applies to `NSArchiver/NSUnarchiver`, but not to `NSKeyedArchiver/NSKeyedUnarchiver`. A keyed archiver does not encode class version numbers.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ version` (page 796)

Declared In

`NSObject.h`

superclass

Returns the class object for the receiver's superclass.

```
+ (Class)superclass
```

Return Value
The class object for the receiver's superclass.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ class` (page 785)
`superclass` (page 1313) (`NSObject` protocol)

Declared In
`NSObject.h`

version

Returns the version number assigned to the class.

```
+ (NSInteger)version
```

Return Value
The version number assigned to the class.

Discussion
If no version has been set, the default is `0`.

Version numbers are needed for decoding or unarchiving, so older versions of an object can be detected and decoded correctly.

Caution should be taken when obtaining the version from within an `NSCoding` protocol or other methods. Use the class name explicitly when getting a class version number:

```
version = [MyClass version];
```

Don't simply send `version` to the return value of class—a subclass version number may be returned instead.

Special Considerations

The version number applies to `NSArchiver/NSUnarchiver`, but not to `NSKeyedArchiver/NSKeyedUnarchiver`. A keyed archiver does not encode class version numbers.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ setVersion:` (page 795)
`versionForClassName:` (page 167) (`NSCoder`)

Declared In
`NSObject.h`

Instance Methods

awakeAfterUsingCoder:

Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.

```
- (id)awakeAfterUsingCoder:(NSCoder *)aDecoder
```

Parameters

aDecoder

The decoder used to decode the receiver.

Return Value

The receiver, or another object to take the place of the object that was decoded and subsequently received this message.

Discussion

This method can be used to eliminate redundant objects created by the coder. For example, if after decoding an object you discover that an equivalent object already exists, you can return the existing object. If a replacement is returned, your overriding method is responsible for releasing the receiver. To prevent the accidental use of the receiver after its replacement has been returned, you should invoke the receiver's `release` method to release the object immediately.

This method is invoked by `NSCoder`. `NSObject`'s implementation simply returns `self`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- classForCoder` (page 797)
`- replacementObjectForCoder:` (page 813)
`initWithCoder:` (page 1246) (`NSCoding` protocol)

Declared In

`NSObject.h`

classForCoder

Overridden by subclasses to substitute a class other than its own during coding.

```
- (Class)classForCoder
```

Return Value

The class to substitute for the receiver's own class during coding.

Discussion

This method is invoked by `NSCoder`. `NSObject`'s implementation returns the receiver's class. The private subclasses of a class cluster substitute the name of their public superclass when being archived.

Availability

Available in iPhone OS 2.0 and later.

See Also
- awakeAfterUsingCoder: (page 797)
- replacementObjectForCoder: (page 813)

Declared In
NSObject.h

classForKeyedArchiver

Overridden by subclasses to substitute a new class for instances during keyed archiving.

```
- (Class)classForKeyedArchiver
```

Discussion
The object will be encoded as if it were a member of the returned class. The results of this method are overridden by the encoder class and instance name to class encoding tables. If nil is returned, the result of this method is ignored.

Availability
Available in iPhone OS 2.0 and later.

See Also
- replacementObjectForKeyedArchiver: (page 814)

Declared In
NSKeyedArchiver.h

copy

Returns the object returned by copyWithZone: (page 1250), where the zone is nil.

```
- (id)copy
```

Return Value
The object returned by the NSCopying protocol method copyWithZone: (page 1250), where the zone is nil.

Discussion
This is a convenience method for classes that adopt the NSCopying protocol. An exception is raised if there is no implementation for copyWithZone: (page 1250).

NSObject does not itself support the NSCopying protocol. Subclasses must support the protocol and implement the copyWithZone: (page 1250) method. A subclass version of the copyWithZone: (page 1250) method should send the message to super first, to incorporate its implementation, unless the subclass descends directly from NSObject.

Special Considerations

If you are using managed memory (not garbage collection), this method retains the new object before returning it. The invoker of the method, however, is responsible for releasing the returned object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

dealloc

Deallocates the memory occupied by the receiver.

```
- (void)dealloc
```

Discussion
Subsequent messages to the receiver will generate an error indicating that a message was sent to a deallocated object (provided the deallocated memory hasn't been reused yet).

You never send a `dealloc` message directly. Instead, an object's `dealloc` method is invoked indirectly through the `release` (page 1310) `NSObject` protocol method (if the `release` message results in the receiver's retain count becoming `0`). See *Memory Management Programming Guide for Cocoa* for more details on the use of these methods.

Subclasses must implement their own versions of `dealloc` to allow the release of any additional memory consumed by the object—such as dynamically allocated storage for data or object instance variables owned by the deallocated object. After performing the class-specific deallocation, the subclass method should incorporate superclass versions of `dealloc` through a message to `super`:

```
- (void)dealloc {
    [companion release];
    NSZoneFree(private, [self zone])
    [super dealloc];
}
```

Note that when an application terminates, objects may not be sent a `dealloc` message since the process's memory is automatically cleared on exit—it is more efficient simply to allow the operating system to clean up resources than to invoke all the memory management methods.

Special Considerations

When garbage collection is enabled, the garbage collector sends `finalize` (page 800) to the receiver instead of `dealloc`.

When garbage collection is enabled, this method is a no-op.

Availability
Available in iPhone OS 2.0 and later.

See Also
`autorelease` (page 1303) (`NSObject` protocol)
`release` (page 1310) (`NSObject` protocol)
`- finalize` (page 800)

Declared In
NSObject.h

doesNotRecognizeSelector:

Handles messages the receiver doesn't recognize.

```
- (void)doesNotRecognizeSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies a method not implemented or recognized by the receiver.

See "Selectors" (page 778) for a description of the `SEL` type.

Discussion

The runtime system invokes this method whenever an object receives an *aSelector* message it can't respond to or forward. This method, in turn, raises an `NSInvalidArgumentException`, and generates an error message.

Any `doesNotRecognizeSelector:` messages are generally sent only by the runtime system. However, they can be used in program code to prevent a method from being inherited. For example, an `NSObject` subclass might renounce the `copy` (page 798) or `init` (page 803) method by re-implementing it to include a `doesNotRecognizeSelector:` message as follows:

```
- copy
{
    [self doesNotRecognizeSelector:_cmd];
}
```

The `_cmd` variable is a hidden argument passed to every method that is the current selector; in this example, it identifies the selector for the `copy` method. This code prevents instances of the subclass from responding to `copy` messages or superclasses from forwarding `copy` messages—although `respondsToSelector:` (page 1311) will still report that the receiver has access to a `copy` method.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- forwardInvocation:` (page 801)

Declared In

`NSObject.h`

finalize

The garbage collector invokes this method on the receiver before disposing of the memory it uses.

```
- (void)finalize
```

Discussion

The garbage collector invokes this method on the receiver before disposing of the memory it uses. When garbage collection is enabled, this method is invoked instead of `dealloc`.

> **Note:** Garbage collection is not available for use in Mac OS X before version 10.5.

You can override this method to relinquish resources the receiver has obtained, as shown in the following example:

```
- (void)finalize {
    if (log_file != NULL) {
        fclose(log_file);
```

```
        log_file = NULL;
    }
    [super finalize];
}
```

Typically, however, you are encouraged to relinquish resources prior to finalization if at all possible. For more details, see Implementing a finalize Method.

Special Considerations

It is an error to store `self` into a new or existing live object (colloquially known as "resurrection"), which implies that this method will be called only once. However, the receiver may be messaged after finalization by other objects also being finalized at this time, so your override should guard against future use of resources that have been reclaimed, as shown by the `log_file = NULL` statement in the example. The `finalize` method itself will never be invoked more than once for a given object.

Important: `finalize` methods must be thread-safe.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- dealloc` (page 799)

Declared In

`NSObject.h`

forwardInvocation:

Overridden by subclasses to forward messages to other objects.

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Parameters

anInvocation

The invocation to forward.

Discussion

When an object is sent a message for which it has no corresponding method, the runtime system gives the receiver an opportunity to delegate the message to another receiver. It delegates the message by creating an `NSInvocation` object representing the message and sending the receiver a `forwardInvocation:` message containing this `NSInvocation` object as the argument. The receiver's `forwardInvocation:` method can then choose to forward the message to another object. (If that object can't respond to the message either, it too will be given a chance to forward it.)

The `forwardInvocation:` message thus allows an object to establish relationships with other objects that will, for certain messages, act on its behalf. The forwarding object is, in a sense, able to "inherit" some of the characteristics of the object it forwards the message to.

Important: To respond to methods that your object does not itself recognize, you must override `methodSignatureForSelector:` (page 805) in addition to `forwardInvocation:`. The mechanism for forwarding messages uses information obtained from `methodSignatureForSelector:` (page 805) to create the `NSInvocation` object to be forwarded. Your overriding method must provide an appropriate method signature for the given selector, either by preformulating one or by asking another object for one.

An implementation of the `forwardInvocation:` method has two tasks:

- To locate an object that can respond to the message encoded in *`anInvocation`*. This object need not be the same for all messages.
- To send the message to that object using *`anInvocation`*. *`anInvocation`* will hold the result, and the runtime system will extract and deliver this result to the original sender.

In the simple case, in which an object forwards messages to just one destination (such as the hypothetical `friend` instance variable in the example below), a `forwardInvocation:` method could be as simple as this:

```
- (void)forwardInvocation:(NSInvocation *)invocation
{
    SEL aSelector = [invocation selector];

    if ([friend respondsToSelector:aSelector])
        [invocation invokeWithTarget:friend];
    else
        [self doesNotRecognizeSelector:aSelector];
}
```

The message that's forwarded must have a fixed number of arguments; variable numbers of arguments (in the style of `printf()`) are not supported.

The return value of the forwarded message is returned to the original sender. All types of return values can be delivered to the sender: `id` types, structures, double-precision floating-point numbers.

Implementations of the `forwardInvocation:` method can do more than just forward messages. `forwardInvocation:` can, for example, be used to consolidate code that responds to a variety of different messages, thus avoiding the necessity of having to write a separate method for each selector. A `forwardInvocation:` method might also involve several other objects in the response to a given message, rather than forward it to just one.

`NSObject`'s implementation of `forwardInvocation:` simply invokes the `doesNotRecognizeSelector:` (page 799) method; it doesn't forward any messages. Thus, if you choose not to implement `forwardInvocation:`, sending unrecognized messages to objects will raise exceptions.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSObject.h`

init

Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

```
- (id)init
```

Return Value

The initialized receiver.

Discussion

An `init` message is generally coupled with an `alloc` (page 783) or `allocWithZone:` (page 783) message in the same line of code:

```
TheClass *newObject = [[TheClass alloc] init];
```

An object isn't ready to be used until it has been initialized. The `init` method defined in the `NSObject` class does no initialization; it simply returns `self`.

Subclass implementations of this method should initialize and return the new object. If it can't be initialized, they should release the object and return `nil`. In some cases, an `init` method might release the new object and return a substitute. Programs should therefore always use the object returned by `init`, and not necessarily the one returned by `alloc` (page 783) or `allocWithZone:` (page 783), in subsequent code.

Every class must guarantee that the `init` method either returns a fully functional instance of the class or raises an exception. Subclasses should override the `init` method to add class-specific initialization code. Subclass versions of `init` need to incorporate the initialization code for the classes they inherit from, through a message to `super`:

```
- init
{
    if ((self = [super init])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

Note that the message to `super` precedes the initialization code added in the method. This sequencing ensures that initialization proceeds in the order of inheritance.

Subclasses often define `init...` methods with additional arguments to allow specific values to be set. The more arguments a method has, the more freedom it gives you to determine the character of initialized objects. Classes often have a set of `init...` methods, each with a different number of arguments. For example:

```
- init;
- initWithTag:(int)tag;
- initWithTag:(int)tag data:(struct info *)data;
```

The convention is that at least one of these methods, usually the one with the most arguments, includes a message to `super` to incorporate the initialization of classes higher up the hierarchy. This method is called the *designated initializer* for the class. The other `init...` methods defined in the class directly or indirectly invoke the designated initializer through messages to `self`. In this way, all `init...` methods are chained together. For example:

```
- init
```

```
{
    return [self initWithTag:-1];
}

- initWithTag:(int)tag
{
    return [self initWithTag:tag data:NULL];
}

- initWithTag:(int)tag data:(struct info *)data
{
    if ((self = [super init. . .])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

In this example, the `initWithTag:data:` method is the designated initializer for the class.

If a subclass does any initialization of its own, it must define its own designated initializer. This method should begin by sending a message to `super` to invoke the designated initializer of its superclass. Suppose, for example, that the three methods illustrated above are defined in the B class. The C class, a subclass of B, might have this designated initializer:

```
- initWithTag:(int)tag data:(struct info *)data data:anObject
{
    if ((self = [super initWithTag:tag data:data])) {
        /* class-specific initialization goes here */
    }
    return self;
}
```

If inherited `init...` methods are to successfully initialize instances of the subclass, they must all be made to (directly or indirectly) invoke the new designated initializer. To accomplish this, the subclass is obliged to cover (override) only the designated initializer of the superclass. For example, in addition to its designated initializer, the C class would also implement this method:

```
- initWithTag:(int)tag data:(struct info *)data
{
    return [self initWithTag:tag data:data arg:nil];
}
```

This code ensures that all three methods inherited from the B class also work for instances of the C class.

Often the designated initializer of the subclass overrides the designated initializer of the superclass. If so, the subclass need only implement the one `init...` method.

These conventions maintain a direct chain of `init...` links and ensure that the `new` method and all inherited `init...` methods return usable, initialized objects. They also prevent the possibility of an infinite loop wherein a subclass method sends a message (to `super`) to perform a superclass method, which in turn sends a message (to `self`) to perform the subclass method.

This `init` method is the designated initializer for the `NSObject` class. Subclasses that do their own initialization should override it, as described above.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

methodForSelector:

Locates and returns the address of the receiver's implementation of a method so it can be called as a function.

```
- (IMP)methodForSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies the method for which to return the implementation address. The selector must be a valid and non-NULL. If in doubt, use the respondsToSelector: (page 1311) method to check before passing the selector to methodForSelector:.

Return Value
The address of the receiver's implementation of the *aSelector*.

Discussion
If the receiver is an instance, *aSelector* should refer to an instance method; if the receiver is a class, it should refer to a class method.

See "Selectors" (page 778) for a description of the IMP and SEL types, and how to invoke the returned method implementation.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ instanceMethodForSelector: (page 790)

Declared In
NSObject.h

methodSignatureForSelector:

Returns an NSMethodSignature object that contains a description of the method identified by a given selector.

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies the method for which to return the implementation address. When the receiver is an instance, *aSelector* should identify an instance method; when the receiver is a class, it should identify a class method.

See "Selectors" (page 778) for a description of the SEL type.

Return Value
An `NSMethodSignature` object that contains a description of the method identified by *`aSelector`*, or `nil` if the method can't be found.

Discussion
This method is used in the implementation of protocols. This method is also used in situations where an `NSInvocation` object must be created, such as during message forwarding. If your object maintains a delegate or is capable of handling messages that it does not directly implement, you should override this method to return an appropriate method signature.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ instanceMethodSignatureForSelector:` (page 790)
`- forwardInvocation:` (page 801)

Declared In
`NSObject.h`

mutableCopy
Returns the object returned by `mutableCopyWithZone:` (page 1300) where the zone is `nil`.

```
- (id)mutableCopy
```

Return Value
The object returned by the `NSMutableCopying` protocol method `mutableCopyWithZone:` (page 1300), where the zone is `nil`.

Discussion
This is a convenience method for classes that adopt the `NSMutableCopying` protocol. An exception is raised if there is no implementation for `mutableCopyWithZone:` (page 1300).

Special Considerations
If you are using managed memory (not garbage collection), this method retains the new object before returning it. The invoker of the method, however, is responsible for releasing the returned object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSObject.h`

performSelector:onThread:withObject:waitUntilDone:
Invokes a method of the receiver on the specified thread using the default mode.

```
- (void)performSelector:(SEL)aSelector onThread:(NSThread *)thr withObject:(id)arg
    waitUntilDone:(BOOL)wait
```

Parameters

aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

thr

The thread on which to execute *aSelector*.

arg

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

wait

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the specified thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread and target thread are the same, and you specify `YES` for this parameter, the selector is performed immediately on the current thread. If you specify `NO`, this method queues the message on the thread's run loop and returns, just like it does for other threads. The current thread must then dequeue and process the message when it has an opportunity to do so.

Discussion

You can use this method to deliver messages to other threads in your application. The message in this case is a method of the current object that you want to execute on the target thread.

This method queues the message on the run loop of the target thread using the default run loop modes—that is, the modes associated with the `NSRunLoopCommonModes` constant. As part of its normal run loop processing, the target thread dequeues the message (assuming it is running in one of the default run loop modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 808) or `performSelector:withObject:afterDelay:inModes:` (page 809) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)
- `performSelectorInBackground:withObject:` (page 810)

Declared In

`NSThread.h`

performSelector:onThread:withObject:waitUntilDone:modes:

Invokes a method of the receiver on the specified thread using the specified modes.

```
- (void)performSelector:(SEL)aSelector onThread:(NSThread *)thr withObject:(id)arg
    waitUntilDone:(BOOL)wait modes:(NSArray *)array
```

Parameters

aSelector

A selector that identifies the method to invoke. It should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

thr

The thread on which to execute *aSelector*. This thread represents the target thread.

arg

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

wait

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the specified thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread and target thread are the same, and you specify `YES` for this parameter, the selector is performed immediately. If you specify `NO`, this method queues the message and returns immediately, regardless of whether the threads are the same or different.

array

An array of strings that identifies the modes in which it is permissible to perform the specified selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector.

Discussion

You can use this method to deliver messages to other threads in your application. The message in this case is a method of the current object that you want to execute on the target thread.

This method queues the message on the run loop of the target thread using the run loop modes specified in the *array* parameter. As part of its normal run loop processing, the target thread dequeues the message (assuming it is running in one of the specified modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 808) or `performSelector:withObject:afterDelay:inModes:` (page 809) method instead.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `performSelector:onThread:withObject:waitUntilDone:` (page 806)
- `performSelectorInBackground:withObject:` (page 810)

Declared In

`NSThread.h`

performSelector:withObject:afterDelay:

Invokes a method of the receiver on the current thread using the default mode after a delay.

```
- (void)performSelector:(SEL)aSelector withObject:(id)anArgument
    afterDelay:(NSTimeInterval)delay
```

Parameters

aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

anArgument

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

delay

The minimum time before which the message is sent.

Discussion

This method sets up a timer to perform the *aSelector* message on the current thread's run loop. The timer is configured to run in the default mode (`NSDefaultRunLoopMode`). When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in the default mode; otherwise, the timer waits until the run loop is in the default mode.

If you want the message to be dequeued when the run loop is in a mode other than the default mode, use the `performSelector:withObject:afterDelay:inModes:` (page 809) method instead. To ensure that the selector is performed on the main thread, use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 784) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 785) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ `cancelPreviousPerformRequestsWithTarget:selector:object:` (page 785)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)

Declared In

`NSRunLoop.h`

performSelector:withObject:afterDelay:inModes:

Invokes a method of the receiver on the current thread using the specified modes after a delay.

```
- (void)performSelector:(SEL)aSelector withObject:(id)anArgument
    afterDelay:(NSTimeInterval)delay inModes:(NSArray *)modes
```

Parameters

aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

anArgument

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

delay

The minimum time before which the message is sent.

modes

An array of strings that identify the modes to associate with the timer that performs the selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector.

Discussion

This method sets up a timer to perform the *aSelector* message on the current thread's run loop. The timer is configured to run in the modes specified by the *modes* parameter. When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in one of the specified modes; otherwise, the timer waits until the run loop is in one of those modes.

If you want the message to be dequeued when the run loop is in a mode other than the default mode, use the `performSelector:withObject:afterDelay:inModes:` (page 809) method instead. To ensure that the selector is performed on the main thread, use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 784) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 785) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 808)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)

`addTimer:forMode:` (page 891) (`NSRunLoop`)

`invalidate` (page 1070) (`NSTimer`)

Declared In

`NSRunLoop.h`

performSelectorInBackground:withObject:

Invokes a method of the receiver on a new background thread.

```
- (void)performSelectorInBackground:(SEL)aSelector withObject:(id)arg
```

Parameters

aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

arg

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

Discussion

This method creates a new thread in your application, putting your application into multithreaded mode if it was not already. The method represented by *aSelector* must set up the thread environment just as you would for any other new thread in your program. For more information about how to configure and run threads, see *Threading Programming Guide*.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)

Declared In

`NSThread.h`

performSelectorOnMainThread:withObject:waitUntilDone:

Invokes a method of the receiver on the main thread using the default mode.

```
- (void)performSelectorOnMainThread:(SEL)aSelector withObject:(id)arg
    waitUntilDone:(BOOL)wait
```

Parameters

aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

arg

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

wait

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the main thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread is also the main thread, and you specify `YES` for this parameter, the message is delivered and processed immediately.

Discussion

You can use this method to deliver messages to the main thread of your application. The main thread encompasses the application's main run loop, and is where the `NSApplication` object receives events. The message in this case is a method of the current object that you want to execute on the thread.

This method queues the message on the run loop of the main thread using the default run loop modes—that is, the modes associated with the `NSRunLoopCommonModes` constant. As part of its normal run loop processing, the main thread dequeues the message (assuming it is running in one of the default run loop modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 808) or `performSelector:withObject:afterDelay:inModes:` (page 809) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `performSelector:withObject:afterDelay:` (page 808)
- `performSelector:withObject:afterDelay:inModes:` (page 809)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 812)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)

Declared In
`NSThread.h`

performSelectorOnMainThread:withObject:waitUntilDone:modes:
Invokes a method of the receiver on the main thread using the specified modes.

```
- (void)performSelectorOnMainThread:(SEL)aSelector withObject:(id)arg
    waitUntilDone:(BOOL)wait modes:(NSArray *)array
```

Parameters
aSelector

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

See "Selectors" (page 778) for a description of the `SEL` type.

arg

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

wait

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the main thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread is also the main thread, and you pass `YES`, the message is performed immediately, otherwise the perform is queued to run the next time through the run loop.

array

An array of strings that identifies the modes in which it is permissible to perform the specified selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector.

Discussion

You can use this method to deliver messages to the main thread of your application. The main thread encompasses the application's main run loop, and is where the `NSApplication` object receives events. The message in this case is a method of the current object that you want to execute on the thread.

This method queues the message on the run loop of the main thread using the run loop modes specified in the *array* parameter. As part of its normal run loop processing, the main thread dequeues the message (assuming it is running in one of the specified modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 808) or `performSelector:withObject:afterDelay:inModes:` (page 809) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 808)
- `performSelector:withObject:afterDelay:inModes:` (page 809)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 811)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 807)

Declared In

`NSThread.h`

replacementObjectForCoder:

Overridden by subclasses to substitute another object for itself during encoding.

```
- (id)replacementObjectForCoder:(NSCoder *)aCoder
```

Parameters

aCoder

The coder encoding the receiver.

Return Value

The object encode instead of the receiver (if different).

Discussion

An object might encode itself into an archive, but encode a proxy for itself if it's being encoded for distribution. This method is invoked by `NSCoder`. `NSObject`'s implementation returns `self`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `classForCoder` (page 797)
- `awakeAfterUsingCoder:` (page 797)

Declared In

`NSObject.h`

replacementObjectForKeyedArchiver:

Overridden by subclasses to substitute another object for itself during keyed archiving.

```
- (id)replacementObjectForKeyedArchiver:(NSKeyedArchiver *)archiver
```

Parameters

archiver

A keyed archiver creating an archive.

Return Value

The object encode instead of the receiver (if different).

Discussion

This method is called only if no replacement mapping for the object has been set up in the encoder (for example, due to a previous call of `replacementObjectForKeyedArchiver:` to that object).

Availability

Available in iPhone OS 2.0 and later.

See Also

`- classForKeyedArchiver` (page 798)

Declared In

`NSKeyedArchiver.h`

NSOperation Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSOperation.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSOperation` class manages the execution of a single encapsulated task. Operations are typically scheduled by adding them to an operation queue object (an instance of the `NSOperationQueue` class), although you can also execute them directly by explicitly invoking their `start` method.

Operation objects are single-shot objects, that is, they perform their task once. You cannot reuse the same `NSOperation` object to perform a task (or a slight variant of the task) multiple times in succession. Attempting to execute an operation that has already finished results in an exception.

When manually executing operations, you are responsible for making sure the object is ready to execute. Starting an operation that is not in the ready state generally results in an exception being thrown. If you use an operation queue to manage the execution, the `NSOperationQueue` object ensures that the operation is executed only when it is ready.

Concurrent Versus Non-Concurrent Operations

Operation objects can be designed for either concurrent or non-concurrent operation. In the context of an `NSOperation` object, the terms concurrent and non-concurrent do not necessarily refer to the side-by-side execution of threads. Instead, a non-concurrent operation is one that executes using the environment that is provided for it while a concurrent operation is responsible for setting up its own execution environment. To understand how this might work in your code, look at the `NSOperationQueue` object as an example. For a non-concurrent operation, an operation queue automatically creates a thread and calls the operation object's `start` method, the default implementation of which configures the thread environment and calls the operation object's `main` method to run your custom code. For a concurrent operation, the queue simply calls `start` without setting up a thread. The operation object is expected to provide a custom implementation of the `start` method that configures the runtime environment of the operation itself.

If you always design your operations to execute on a thread, then creating non-concurrent operations is the simplest way to go. There are some situations though where you might want to create a concurrent operation instead, including the following:

- You want to create the thread yourself.
- You want to launch a separate task instead of a thread.
- Your operation's main method initiates an asynchronous call and exits. (In such a situation, the callback function or method would then pass control to the operation object to process the request. For example, you could use this technique to set up a timer and then use the methods of the operation object to do some work each time the timer fires.)

By default, operations are designated as non-concurrent. For information on how to create a concurrent operation object, see the subclassing notes for this class.

Operation Dependencies

You can configure an operation to depend on the completion of other operations by adding those operations as dependencies. An operation object that has dependencies does not execute until all of its dependent operation objects finish executing. Once the last dependent operation finishes, the operation object moves to the ready state.

If a dependent operation is unable to perform its task for some reason, it is the responsibility of your code to make that determination. Operation objects that are non-concurrent (that is, their `isConcurrent` method returns `NO`) automatically catch and suppress any exceptions thrown by the operation object's `main` method. Thus, an operation that generates an exception may appear to finish normally even if it did not. If you need to track errors in a dependent operation, you must build that capability into the `main` method of your operation objects explicitly.

Bindable Properties

The `NSOperation` class is key-value coding (KVC) and key-value observing (KVO) compliant for several of its properties. As needed, you can establish bindings to these properties to control other parts of your application. The properties you can bind to include the following:

- `isCancelled`- read-only property
- `isConcurrent`- read-only property
- `isExecuting` - read-only property
- `isFinished` - read-only property
- `isReady` - read-only property
- `dependencies` - read-only property
- `queuePriority` - readable and writable property

If you override any of the preceding properties, your implementations must maintain KVC and KVO compliance. If you define additional properties for your `NSOperation` objects, it is recommended that you make those properties KVC and KVO compliant as well. For information on how to support key-value coding, see *Key-Value Coding Programming Guide*. For information on how to support key-value observing, see *Key-Value Observing Programming Guide*.

Threading Considerations

The methods of the `NSOperation` class implement automatic synchronization on the current instance. It is therefore safe to use a single instance of the `NSOperation` object from multiple threads without creating additional locks to synchronize access to the object.

When you subclass `NSOperation`, the methods in your implementation should also be safe to call from multiple threads. One way to achieve this is to synchronize access to the operation object using the `@synchronized` keyword. For more information about writing thread-safe code, see *Threading Programming Guide*.

Subclassing Notes

The `NSOperation` class does not do anything by default and must be subclassed to perform any desired tasks. How you create your subclass depends on whether your operation is designed to execute concurrently or non-concurrently with respect to the thread that started the operation.

Methods to Override

For non-concurrent operations, you typically implement only one method:

- `main`

In your `main` method, you implement the code needed to perform the given operation. The `NSOperation` class manages the changes in state for your operation automatically and reports the appropriate condition of your operation from its methods.

If you are creating a concurrent operation, you need to override the following methods:

- `start`
- `isConcurrent`
- `isExecuting`
- `isFinished`

In your `start` method, you must prepare the operation for execution, which includes preparing the runtime environment for your operation. (For example, if you wanted to create a thread yourself, you would do it here.) Once your runtime environment is established, you can call any methods or functions you want to subsequently start your operation. Your implementation of the `start` method should not invoke `super`.

When implementing a concurrent operation, your custom subclass is responsible for reporting some of the state information associated with running the operation. In particular, you must override the `isExecuting` and `isFinished` methods to report on the current execution state of your operation. Your overridden methods should be KVO compliant.

Responding to the Cancel Command

An operation is responsible for periodically calling its own `isCancelled` method and aborting execution if it ever returns `YES`. Because it is bad form to kill a thread outright, the `NSOperationQueue` object sends a `cancel` message to your operation object if it ever needs your object to stop executing. (Other entities can similarly call the `cancel` method on an executing operation to ask it to stop.) The need to cancel an operation can typically arise from a user request or in a situation where the application or system is shutting down. When detected, your operation should clean up its environment and exit as soon as possible.

Tasks

Initialization

`- init` (page 821)
Returns an initialized `NSOperation` object.

Executing the Operation

`- start` (page 825)
Begins the execution of the operation.

`- main` (page 823)
Performs the receiver's non-concurrent task.

Canceling Operations

- `cancel` (page 820)
  Advises the operation object that it should stop executing its task.

Getting the Operation Status

- `isCancelled` (page 821)
  Returns a Boolean value indicating whether the operation has been cancelled.
- `isExecuting` (page 822)
  Returns a Boolean value indicating whether the operation is currently executing.
- `isFinished` (page 822)
  Returns a Boolean value indicating whether the operation is done executing.
- `isConcurrent` (page 822)
  Returns a Boolean value indicating whether the operation runs asynchronously.
- `isReady` (page 823)
  Returns a Boolean value indicating whether the receiver's operation can be performed now.

Managing Dependencies

- `addDependency:` (page 819)
  Makes the receiver dependent on the completion of the specified operation.
- `removeDependency:` (page 824)
  Removes the receiver's dependence on the specified operation.
- `dependencies` (page 820)
  Returns the operations on which the receiver is dependent.

Prioritizing Operations in an Operation Queue

- `queuePriority` (page 824)
  Returns the priority of the operation in an operation queue.
- `setQueuePriority:` (page 825)
  Sets the priority of the operation when used in an operation queue.

Instance Methods

addDependency:

Makes the receiver dependent on the completion of the specified operation.

```
- (void)addDependency:(NSOperation *)operation
```

Parameters

operation

The operation on which the receiver is dependent. The same dependency should not be added more than once to the receiver, and the results of doing so are undefined.

Discussion

The receiver is not considered ready to execute until all of its dependent operations finish executing. If the receiver is already executing its task, adding dependencies is unlikely to have any practical effect. This method may change the `isReady` and `dependencies` properties of the receiver.

It is a programmer error to create any circular dependencies among a set of operations. Doing so can cause a deadlock among the operations and may freeze your program.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `removeDependency:` (page 824)
- `dependencies` (page 820)

Declared In

`NSOperation.h`

cancel

Advises the operation object that it should stop executing its task.

```
- (void)cancel
```

Discussion

This method does not force your operation code to stop. The code for your operation must invoke the `isCancelled` method periodically to determine whether the operation should be stopped. Once cancelled, an operation cannot be restarted.

If the operation is already finished executing, this method has no effect. Canceling an operation that is currently in an operation queue, but not yet executing, causes it to be removed from the queue (although not necessarily right away).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isCancelled` (page 821)

Declared In

`NSOperation.h`

dependencies

Returns the operations on which the receiver is dependent.

```
- (NSArray *)dependencies
```

Return Value
An array of `NSOperation` objects representing the operations on which the receiver is dependent.

Discussion
The receiver is not considered ready to execute until all of its dependent operations finish executing.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addDependency:` (page 819)
- `removeDependency:` (page 824)

Declared In
`NSOperation.h`

init

Returns an initialized `NSOperation` object.

```
- (id)init
```

Return Value
The initialized `NSOperation` object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSOperation.h`

isCancelled

Returns a Boolean value indicating whether the operation has been cancelled.

```
- (BOOL)isCancelled
```

Return Value
`YES` if the operation was explicitly cancelled by an invocation of the receiver's `cancel` method; otherwise, `NO`. This method may return `YES` even if the operation is currently executing.

Discussion
Canceling an operation does not actively stop the receiver's code from executing. An operation object is responsible for calling this method periodically and stopping itself if the method returns `YES`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `cancel` (page 820)

Declared In
`NSOperation.h`

isConcurrent

Returns a Boolean value indicating whether the operation runs asynchronously.

```
- (BOOL)isConcurrent
```

Return Value

`YES` if the operation is asynchronous; otherwise, `NO` if the operation runs synchronously on whatever thread started it. This method returns `NO` by default.

Discussion

If you are implementing a concurrent operation, you must override this method and return `YES` from your implementation. For more information about the differences between concurrent and non-concurrent operations, see "Concurrent Versus Non-Concurrent Operations" (page 816).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSOperation.h`

isExecuting

Returns a Boolean value indicating whether the operation is currently executing.

```
- (BOOL)isExecuting
```

Return Value

`YES` if the operation is executing; otherwise, `NO` if the operation has not been started or is already finished.

Discussion

If you are implementing a concurrent operation, you should override this method to return the execution state of your operation. Concurrent operations are also responsible for generating the appropriate KVO notifications whenever the execution state changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSOperation.h`

isFinished

Returns a Boolean value indicating whether the operation is done executing.

```
- (BOOL)isFinished
```

Return Value

`YES` if the operation is no longer executing; otherwise, `NO`.

Discussion
If you are implementing a concurrent operation, you should override this method to return the finished state of your operation. Concurrent operations are also responsible for generating the appropriate KVO notifications whenever the finished state changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSOperation.h`

isReady
Returns a Boolean value indicating whether the receiver's operation can be performed now.

```
- (BOOL)isReady
```

Return Value
YES if the operation can be performed now; otherwise, NO.

Discussion
Operations may not be ready due to dependencies on other operations or because of external conditions that might prevent needed data from being ready. The `NSOperation` class manages dependencies on other operations and reports the readiness of the receiver based on those dependencies.

> **Note:** If the receiver is cancelled before it starts, operations that are dependent on the completion of the receiver will never become ready.

If your operation object has additional dependencies, you must override this method and return a value that accurately reflects the readiness of the receiver. Your custom implementation should invoke `super` and incorporate its return value into this method's return value. Your custom implementation must be KVO compliant.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- dependencies` (page 820)

Declared In
`NSOperation.h`

main
Performs the receiver's non-concurrent task.

```
- (void)main
```

Discussion
The default implementation of this method does nothing. For non-concurrent operations, you must override this method in your `NSOperation` subclass to perform the desired task. In your implementation, do not invoke `super`.

If you are implementing a concurrent operation, you should override the `start` method instead. In your overridden `start` method, you can continue to call this method to do the actual work if separating the work from your starting logic is practical.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- start` (page 825)

Declared In
`NSOperation.h`

queuePriority

Returns the priority of the operation in an operation queue.

```
- (NSOperationQueuePriority)queuePriority
```

Return Value
The relative priority of the operation. The returned value always corresponds to one of the predefined constants. (For a list of valid values, see "Operation Priorities" (page 826).) If no priority is explicitly set, this method returns `NSOperationQueuePriorityNormal`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setQueuePriority:` (page 825)

Declared In
`NSOperation.h`

removeDependency:

Removes the receiver's dependence on the specified operation.

```
- (void)removeDependency:(NSOperation *)operation
```

Parameters
operation
The dependent operation to be removed from the receiver.

Discussion
This method may change the `isReady` and `dependencies` properties of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `addDependency:` (page 819)
- `dependencies` (page 820)

Declared In
`NSOperation.h`

setQueuePriority:

Sets the priority of the operation when used in an operation queue.

```
- (void)setQueuePriority:(NSOperationQueuePriority)priority
```

Parameters

priority

The relative priority of the operation. For a list of valid values, see "Operation Priorities" (page 826).

Discussion
You should use priority values only as needed to classify the relative priority of non-dependent operations. Priority values should not be used to implement dependency management among different operation objects. If you need to establish dependencies between operations, use the `addDependency:` method instead.

If you attempt to specify a priority value that does not match one of the defined constants, this method automatically adjusts the value you specify towards the `NSOperationQueuePriorityNormal` priority, stopping at the first valid constant value. For example, if you specified the value -10, this method would adjust that value to match the `NSOperationQueuePriorityVeryLow` constant. Similarly, if you specified +10, this method would adjust the value to match the `NSOperationQueuePriorityVeryHigh` constant.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `queuePriority` (page 824)
- `addDependency:` (page 819)

Declared In
`NSOperation.h`

start

Begins the execution of the operation.

```
- (void)start
```

Discussion
The default implementation of this method configures the execution environment for a non-concurrent operation and invokes the receiver's `main` method. As part of the default configuration, this method performs several checks to ensure that the non-concurrent operation can actually run and generates appropriate KVO notifications for each change in the operation's state. If the receiver's operation has

already been performed, was cancelled, or is not yet ready to run, this method throws an `NSInvalidArgumentException` exception. If the operation is to be performed on a separate thread, this method may return before the operation itself completes on the other thread.

Note: An operation may not be ready to execute if it is dependent on other operations that have not yet finished.

If you are implementing a concurrent operation, you must override this method to initiate your operation; however, your implementation must not call `super`. If you override this method, you must also override the `isExecuting` and `isFinished` methods to report when your operation begins executing and finishes. Your implementations for these methods must maintain KVO compliance for the associated properties by manually sending the appropriate value change messages. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `main` (page 823)
- `isReady` (page 823)
- `dependencies` (page 820)

Declared In
`NSOperation.h`

Constants

NSOperationQueuePriority

Describes the priority of an operation relative to other operations in an operation queue.

```
typedef NSInteger NSOperationQueuePriority;
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSOperation.h`

Operation Priorities

These constants let you prioritize the order in which operations execute.

```
enum {
    NSOperationQueuePriorityVeryLow = -8,
    NSOperationQueuePriorityLow = -4,
    NSOperationQueuePriorityNormal = 0,
    NSOperationQueuePriorityHigh = 4,
    NSOperationQueuePriorityVeryHigh = 8
};
```

Constants

`NSOperationQueuePriorityVeryLow`
Operations receive very low priority for execution.
Available in iPhone OS 2.0 and later.
Declared in `NSOperation.h`

`NSOperationQueuePriorityLow`
Operations receive low priority for execution.
Available in iPhone OS 2.0 and later.
Declared in `NSOperation.h`

`NSOperationQueuePriorityNormal`
Operations receive the normal priority for execution.
Available in iPhone OS 2.0 and later.
Declared in `NSOperation.h`

`NSOperationQueuePriorityHigh`
Operations receive high priority for execution.
Available in iPhone OS 2.0 and later.
Declared in `NSOperation.h`

`NSOperationQueuePriorityVeryHigh`
Operations receive very high priority for execution.
Available in iPhone OS 2.0 and later.
Declared in `NSOperation.h`

Discussion

You can use these constants to specify the relative ordering of operations that are waiting to be started in an operation queue. You should always use these constants (and not the defined value) for determining priority.

Declared In

`NSOperation.h`

NSOperationQueue Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSOperation.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSOperationQueue` class manages a set of `NSOperation` objects in a priority queue and regulates their execution. Operations remain in the queue until they are explicitly cancelled or finish executing. An application may create multiple operation queues, with each queue running up to its designated maximum number of operations.

A specific `NSOperation` object can be in only one operation queue at a time. Operations within a single queue coordinate their execution order using both priority levels and inter-operation object dependencies. Operation objects in different queues can coordinate their execution order using dependencies, which are not queue-specific.

Inter-operation dependencies provide an absolute execution order for operations. An operation object is not considered ready to execute until all of its dependent operations have finished executing. For operations that are ready to execute, the operation queue always executes the one with the highest priority relative to the other ready operations. For details on how to set priority levels and dependencies, see *NSOperation Class Reference*.

You should never manually start an operation while it is sitting in an operation queue. Once added, an operation stays in its queue until it finishes executing or is cancelled.

If the `isConcurrent` method of an operation returns `NO`, the operation queue automatically creates a new thread for that operation before running it. If the `isConcurrent` method returns `YES`, the operation object must create its own thread or otherwise configure its own runtime environment as part of its execution phase.

Bindable Properties

The `NSOperationQueue` class is key-value coding (KVC) and key-value observing (KVO) compliant. You can establish bindings to these properties as desired to control other parts of your application. The properties you can bind to include the following:

- `operations` - read-only property
- `maxConcurrentOperationCount` - readable and writable property

Threading Considerations

The methods of the `NSOperationQueue` class implement automatic synchronization on the current instance. It is therefore safe to use a single instance of the `NSOperationQueue` object from multiple threads without creating additional locks to synchronize access to the object.

Tasks

Managing Operations in the Queue

- `addOperation:` (page 831)
  Adds the specified operation object to the receiver.
- `operations` (page 833)
  Returns the operations currently in the queue.
- `cancelAllOperations` (page 831)
  Cancels all queued and executing operations.
- `waitUntilAllOperationsAreFinished` (page 834)
  Blocks the current thread until all of the receiver's queued and executing operations finish executing.

Managing the Number of Running Operations

- `maxConcurrentOperationCount` (page 832)
  Returns the maximum number of concurrent operations that the receiver can execute.
- `setMaxConcurrentOperationCount:` (page 833)
  Sets the maximum number of concurrent operations that the receiver can execute.

Suspending Operations

- `setSuspended:` (page 833)
  Modifies the execution of pending operations
- `isSuspended` (page 832)
  Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

Instance Methods

addOperation:

Adds the specified operation object to the receiver.

```
- (void)addOperation:(NSOperation *)operation
```

Parameters

operation
The operation object to be added to the queue.

Discussion

An operation object can be in at most one operation queue at a time and cannot be added if it is currently executing or finished. This method throws an `NSInvalidArgumentException` exception if any of these conditions is true.

Once added, the specified *operation* remains in the queue until it is executed or cancelled.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `cancel` (page 820) (NSOperation)
- `isExecuting` (page 822) (NSOperation)

Declared In

`NSOperation.h`

cancelAllOperations

Cancels all queued and executing operations.

```
- (void)cancelAllOperations
```

Discussion
This method sends a `cancel` message to all operations currently in the queue or executing. Queued operations are cancelled before they begin executing. If an operation is already executing, it is up to that operation to recognize the cancellation and stop what it is doing.

Availability
Available in iPhone OS 2.0 and later.

See Also
`cancel` (page 820) (`NSOperation`)

Declared In
`NSOperation.h`

isSuspended

Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

```
- (BOOL)isSuspended
```

Return Value
`YES` if operations are being scheduled for execution; otherwise, `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setSuspended:` (page 833)

Declared In
`NSOperation.h`

maxConcurrentOperationCount

Returns the maximum number of concurrent operations that the receiver can execute.

```
- (NSInteger)maxConcurrentOperationCount
```

Return Value
The maximum number of concurrent operations set explicitly on the receiver using the `setMaxConcurrentOperationCount:` method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setMaxConcurrentOperationCount:` (page 833)

Declared In
`NSOperation.h`

operations

Returns the operations currently in the queue.

```
- (NSArray *)operations
```

Return Value

An array of `NSOperation` objects in arranged in the order in which they were added to the queue.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSOperation.h`

setMaxConcurrentOperationCount:

Sets the maximum number of concurrent operations that the receiver can execute.

```
- (void)setMaxConcurrentOperationCount:(NSInteger)count
```

Parameters

count

The maximum number of concurrent operations. Specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` if you want the receiver to choose an appropriate value based on the number of available processors and other relevant factors.

Discussion

The specified value affects only the receiver and the operations in its queue. Other operation queue objects can also execute their maximum number of operations in parallel.

Reducing the number of concurrent operations does not affect any operations that are currently executing. If you specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` (which is recommended), the maximum number of operations can change dynamically based on system conditions.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- maxConcurrentOperationCount` (page 832)

Declared In

`NSOperation.h`

setSuspended:

Modifies the execution of pending operations

```
- (void)setSuspended:(BOOL)suspend
```

Parameters

suspend

If YES, the queue stops scheduling queued operations for execution. If NO, the queue begins scheduling operations again.

Discussion

This method suspends or restarts the execution of queued operations only. It does not have any impact on the state of currently running operations. Running operations continue to run until their natural termination or until they are explicitly cancelled.

Availability

Available in iPhone OS 2.0 and later.

See Also

- isSuspended (page 832)

Declared In

NSOperation.h

waitUntilAllOperationsAreFinished

Blocks the current thread until all of the receiver's queued and executing operations finish executing.

```
- (void)waitUntilAllOperationsAreFinished
```

Discussion

When called, this method blocks the current thread and waits for the receiver's current and pending operations to finish executing. While the thread is blocked, the receiver continues to launch already queued operations and monitor those that are executing. During this time, the current thread cannot add operations to the queue, but other threads may. Once all of the pending operations are finished, this method returns.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSOperation.h

Constants

Concurrent Operation Constants

Indicates the number of supported concurrent operations.

```
enum {
    NSOperationQueueDefaultMaxConcurrentOperationCount = -1
};
```

Constants

`NSOperationQueueDefaultMaxConcurrentOperationCount`

The default maximum number of operations is determined dynamically by the `NSOperationQueue` object based on current system conditions.

Available in iPhone OS 2.0 and later.

Declared in `NSOperation.h`

Declared In

`NSOperation.h`

NSOutputStream Class Reference

Inherits from:	NSStream : NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSStream.h
Companion guide:	Stream Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSOutputStream` class is a subclass of `NSStream` that provides write-only stream functionality.

Subclassing Notes

The `NSOutputStream` is a concrete subclass of `NSStream` that lets you write data to a stream. Although `NSOutputStream` is probably sufficient for most situations requiring this capability, you can create a subclass of `NSOutputStream` if you want more specialized behavior (for example, you want to record statistics on the data in a stream).

Methods to Override

To create a subclass of `NSOutputStream` you may have to implement initializers for the type of stream data supported and suitably reimplement existing initializers. You must also provide complete implementations of the following methods:

- `write:maxLength:` (page 842)

  From the current write pointer, take up to the number of bytes specified in the `maxLength:` parameter from the client-supplied buffer (first parameter) and put them onto the stream. The buffer must be of the size specified by the second parameter. To prepare for the next operation, offset the write pointer by the number of bytes written. Return a signed integer based on the outcome of the current operation:

  - If the write operation is successful, return the actual number of bytes put onto the stream.
  - If there was an error writing to the stream, return -1.
  - If the stream is of a fixed length and has reached its capacity, return zero.

- `hasSpaceAvailable` (page 840)

  Return `YES` if the stream can currently accept more data, `NO` if it cannot. If you want to be semantically compatible with `NSOutputStream`, return `YES` if a write must be attempted to determine if space is available.

Tasks

Creating Streams

`+ outputStreamToMemory` (page 840)
Creates and returns an initialized output stream that will write stream data to memory.

`+ outputStreamToBuffer:capacity:` (page 839)
Creates and returns an initialized output stream that can write to a provided buffer.

`+ outputStreamToFileAtPath:append:` (page 839)
Creates and returns an initialized output stream for writing to a specified file.

`- initToMemory` (page 842)
Returns an initialized output stream that will write to memory.

`- initToBuffer:capacity:` (page 841)
Returns an initialized output stream that can write to a provided buffer.

`- initToFileAtPath:append:` (page 841)
Returns an initialized output stream for writing to a specified file.

Using Streams

`- hasSpaceAvailable` (page 840)
Returns whether the receiver can be written to.

- write:maxLength: (page 842)
  Writes the contents of a provided data buffer to the receiver.

Class Methods

outputStreamToBuffer:capacity:

Creates and returns an initialized output stream that can write to a provided buffer.

```
+ (id)outputStreamToBuffer:(uint8_t *)buffer capacity:(NSUInteger)capacity
```

Parameters

buffer
  The buffer the output stream will write to.

capacity
  The size of the buffer in bytes.

Return Value

An initialized output stream that can write to *buffer*.

Discussion

The stream must be opened before it can be used.

When the number of bytes written to *buffer* has reached *capacity*, the stream's streamStatus (page 949) will return NSStreamStatusAtEnd.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ outputStreamToMemory (page 840)
+ outputStreamToFileAtPath:append: (page 839)
- initToBuffer:capacity: (page 841)

Declared In

NSStream.h

outputStreamToFileAtPath:append:

Creates and returns an initialized output stream for writing to a specified file.

```
+ (id)outputStreamToFileAtPath:(NSString *)path append:(BOOL)shouldAppend
```

Parameters

path
  The path to the file the output stream will write to.

shouldAppend
  YES if newly written data should be appended to any existing file contents, NO otherwise.

Return Value
An initialized output stream that can write to *path*.

Discussion
The stream must be opened before it can be used.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ outputStreamToMemory (page 840)
+ outputStreamToBuffer:capacity: (page 839)
- initToFileAtPath:append: (page 841)

Declared In
NSStream.h

outputStreamToMemory

Creates and returns an initialized output stream that will write stream data to memory.

```
+ (id)outputStreamToMemory
```

Return Value
An initialized output stream that will write stream data to memory.

Discussion
The stream must be opened before it can be used.

You retrieve the contents of the memory stream by sending the message propertyForKey: (page 947) to the receiver with an argument of NSStreamDataWrittenToMemoryStreamKey.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ outputStreamToBuffer:capacity: (page 839)
+ outputStreamToFileAtPath:append: (page 839)
- initToMemory (page 842)

Declared In
NSStream.h

Instance Methods

hasSpaceAvailable

Returns whether the receiver can be written to.

```
- (BOOL)hasSpaceAvailable
```

Return Value
YES if the receiver can be written to or if a write must be attempted in order to determine if space is available, NO otherwise.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSStream.h

initToBuffer:capacity:

Returns an initialized output stream that can write to a provided buffer.

```
- (id)initToBuffer:(uint8_t *)buffer capacity:(NSUInteger)capacity
```

Parameters
buffer
The buffer the output stream will write to.

capacity
The size of the buffer in bytes.

Return Value
An initialized output stream that can write to *buffer*.

Discussion
The stream must be opened before it can be used.

When the number of bytes written to *buffer* has reached *capacity*, the stream's streamStatus (page 949) will return NSStreamStatusAtEnd.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initToMemory (page 842)
- initToFileAtPath:append: (page 841)
+ outputStreamToBuffer:capacity: (page 839)

Declared In
NSStream.h

initToFileAtPath:append:

Returns an initialized output stream for writing to a specified file.

```
- (id)initToFileAtPath:(NSString *)path append:(BOOL)shouldAppend
```

Parameters
path
The path to the file the output stream will write to.

shouldAppend
YES if newly written data should be appended to any existing file contents, NO otherwise.

Return Value
An initialized output stream that can write to *path*.

Discussion
The stream must be opened before it can be used.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initToMemory (page 842)
- initToBuffer:capacity: (page 841)
+ outputStreamToFileAtPath:append: (page 839)

Declared In
NSStream.h

initToMemory

Returns an initialized output stream that will write to memory.

```
- (id)initToMemory
```

Return Value
An initialized output stream that will write stream data to memory.

Discussion
The stream must be opened before it can be used.

The contents of the memory stream are retrieved by passing the constant NSStreamDataWrittenToMemoryStreamKey to propertyForKey: (page 947).

Availability
Available in iPhone OS 2.0 and later.

See Also
- initToBuffer:capacity: (page 841)
- initToFileAtPath:append: (page 841)
+ outputStreamToMemory (page 840)

Declared In
NSStream.h

write:maxLength:

Writes the contents of a provided data buffer to the receiver.

```
- (NSInteger)write:(const uint8_t *)buffer maxLength:(NSUInteger)length
```

Parameters

buffer

The data to write.

length

The length of the data buffer, in bytes.

Return Value

The number of bytes actually written, or `-1` if an error occurs. If the receiver is a fixed-length stream and has reached its capacity, `0` is returned.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSStream.h`

NSPipe Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSFileHandle.h
Companion guide:	Interacting with the Operating System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSPipe` objects provide an object-oriented interface for accessing pipes. An `NSPipe` object represents both ends of a pipe and enables communication through the pipe. A pipe is a one-way communications channel between related processes; one process writes data, while the other process reads that data. The data that passes through the pipe is buffered; the size of the buffer is determined by the underlying operating system. `NSPipe` is an abstract class, the public interface of a class cluster.

Tasks

Creating an NSPipe Object

- `init` (page 847)
  Returns an initialized `NSPipe` object.
+ `pipe` (page 846)
  Returns an `NSPipe` object.

Getting the File Handles for a Pipe

- `fileHandleForReading` (page 846)
  Returns the receiver's read file handle.
- `fileHandleForWriting` (page 847)
  Returns the receiver's write file handle.

Class Methods

pipe

Returns an `NSPipe` object.

```
+ (id)pipe
```

Return Value
An initialized `NSPipe` object. Returns `nil` if the method encounters errors while attempting to create the pipe or the `NSFileHandle` objects that serve as endpoints of the pipe.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

Instance Methods

fileHandleForReading

Returns the receiver's read file handle.

```
- (NSFileHandle *)fileHandleForReading
```

Return Value
The receiver's read file handle.The descriptor represented by this object is deleted, and the object itself is automatically deallocated when the receiver is deallocated.

Discussion
You use the returned file handle to read from the pipe using `NSFileHandle`'s read methods—`availableData` (page 368), `readDataToEndOfFile` (page 372), and `readDataOfLength:` (page 371).

You don't need to send `closeFile` (page 369) to this object or explicitly release the object after you have finished using it.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

fileHandleForWriting

Returns the receiver's write file handle.

```
- (NSFileHandle *)fileHandleForWriting
```

Return Value
The receiver's write file handle. This object is automatically deallocated when the receiver is deallocated.

Discussion
You use the returned file handle to write to the pipe using `NSFileHandle`'s `writeData:` (page 377) method. When you are finished writing data to this object, send it a `closeFile` (page 369) message to delete the descriptor. Deleting the descriptor causes the reading process to receive an end-of-data signal (an empty `NSData` object).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSFileHandle.h`

init

Returns an initialized `NSPipe` object.

```
- (id)init
```

Return Value
An initialized `NSPipe` object. Returns `nil` if the method encounters errors while attempting to create the pipe or the `NSFileHandle` objects that serve as endpoints of the pipe.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ pipe` (page 846)

Declared In
NSFileHandle.h

NSPort Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSPort.h
Companion guides:	Run Loops Distributed Objects

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSPort` is an abstract class that represents a communication channel. Communication occurs between `NSPort` objects, which typically reside in different threads or tasks. The distributed objects system uses `NSPort` objects to send `NSPortMessage` objects back and forth. You should implement interapplication communication using distributed objects whenever possible and use `NSPort` objects only when necessary.

To receive incoming messages, `NSPort` objects must be added to an `NSRunLoop` object as input sources. `NSConnection` objects automatically add their receive port when initialized.

When an NSPort object receives a port message, it forwards the message to its delegate in a handleMachMessage: (page 556) or handlePortMessage: (page 857) message. The delegate should implement only one of these methods to process the incoming message in whatever form desired. handleMachMessage: (page 556) provides a message as a raw Mach message beginning with a msg_header_t structure. handlePortMessage: (page 857) provides a message as an NSPortMessage object, which is an object-oriented wrapper for a Mach message. If a delegate has not been set, the NSPort object handles the message itself.

When you are finished using a port object, you must explicitly invalidate the port object prior to sending it a release message. Similarly, if your application uses garbage collection, you must invalidate the port object before removing any strong references to it. If you do not invalidate the port, the resulting port object may linger and create a memory leak. To invalidate the port object, invoke its invalidate method.

Foundation defines three concrete subclasses of NSPort. NSMachPort and NSMessagePort allow local (on the same machine) communication only. NSSocketPort allows for both local and remote communication, but may be more expensive than the others for the local case. When creating an NSPort object, using allocWithZone: (page 852) or port (page 852), an NSMachPort object is created instead.

Important: NSPort conforms to the NSCoding protocol, but only supports coding by an NSPortCoder. NSPort and its subclasses do not support archiving.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

Tasks

Creating Instances

+ allocWithZone: (page 852)
Returns an instance of the NSMachPort class.

+ port (page 852)
Creates and returns a new NSPort object capable of both sending and receiving messages.

Validation

- `invalidate` (page 853)
  Marks the receiver as invalid and posts an `NSPortDidBecomeInvalidNotification` (page 857) to the default notification center.
- `isValid` (page 853)
  Returns a Boolean value that indicates whether the receiver is valid.

Setting the Delegate

- `setDelegate:` (page 856)
  Sets the receiver's delegate to a given object.
- `delegate` (page 853)
  Returns the receiver's delegate.

Setting Information

- `sendBeforeDate:components:from:reserved:` (page 855)
  This method is provided for subclasses that have custom types of `NSPort`.
- `sendBeforeDate:msgid:components:from:reserved:` (page 856)
  This method is provided for subclasses that have custom types of `NSPort`.
- `reservedSpaceLength` (page 854)
  Returns the number of bytes of space reserved by the receiver for sending data.

Port Monitoring

- `removeFromRunLoop:forMode:` (page 854)
  This method should be implemented by a subclass to stop monitoring of a port when removed from a give run loop in a given input mode.
- `scheduleInRunLoop:forMode:` (page 855)
  This method should be implemented by a subclass to set up monitoring of a port when added to a given run loop in a given input mode.

Handling Port Messages

- `handlePortMessage:` (page 857) *delegate method*
  Processes a given incoming message on the port.

Class Methods

allocWithZone:

Returns an instance of the `NSMachPort` class.

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

Parameters

zone

The memory zone in which to allocate the new object.

Return Value

An instance of the `NSMachPort` class.

Discussion

For backward compatibility on Mach, `allocWithZone:` returns an instance of the `NSMachPort` class when sent to the `NSPort` class. Otherwise, it returns an instance of a concrete subclass that can be used for messaging between threads or processes on the local machine, or, in the case of `NSSocketPort`, between processes on separate machines.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

port

Creates and returns a new `NSPort` object capable of both sending and receiving messages.

```
+ (NSPort *)port
```

Return Value

A new `NSPort` object capable of both sending and receiving messages.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ allocWithZone:` (page 852)

Declared In

`NSPort.h`

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setDelegate: (page 856)

Declared In
NSPort.h

invalidate

Marks the receiver as invalid and posts an NSPortDidBecomeInvalidNotification (page 857) to the default notification center.

```
- (void)invalidate
```

Discussion
You must call this method before releasing a port object (or removing strong references to it if your application is garbage collected).

Availability
Available in iPhone OS 2.0 and later.

See Also
- isValid (page 853)

Declared In
NSPort.h

isValid

Returns a Boolean value that indicates whether the receiver is valid.

```
- (BOOL)isValid
```

Return Value
NO if the receiver is known to be invalid, otherwise YES.

Discussion
An NSPort object becomes invalid when its underlying communication resource, which is operating system dependent, is closed or damaged.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `invalidate` (page 853)

Declared In
`NSPort.h`

removeFromRunLoop:forMode:

This method should be implemented by a subclass to stop monitoring of a port when removed from a give run loop in a given input mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop
The run loop from which to remove the receiver.

mode
The run loop mode from which to remove the receiver

Discussion
This method should not be called directly.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `scheduleInRunLoop:forMode:` (page 855)

Declared In
`NSPort.h`

reservedSpaceLength

Returns the number of bytes of space reserved by the receiver for sending data.

```
- (NSUInteger)reservedSpaceLength
```

Return Value
The number of bytes reserved by the receiver for sending data. The default length is `0`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSPort.h`

scheduleInRunLoop:forMode:

This method should be implemented by a subclass to set up monitoring of a port when added to a given run loop in a given input mode.

```
- (void)scheduleInRunLoop:(NSRunLoop *)runLoop forMode:(NSString *)mode
```

Parameters

runLoop
The run loop to which to add the receiver.

mode
The run loop mode to which to add the receiver

Discussion

This method should not be called directly.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeFromRunLoop:forMode: (page 854)

Declared In

NSPort.h

sendBeforeDate:components:from:reserved:

This method is provided for subclasses that have custom types of NSPort.

```
- (BOOL)sendBeforeDate:(NSDate *)limitDate components:(NSMutableArray *)components
    from:(NSPort *)receivePort reserved:(NSUInteger)headerSpaceReserved
```

Parameters

limitDate
The last instant that a message may be sent.

components
The message components.

receivePort
The receive port.

headerSpaceReserved
The number of bytes reserved for the header.

Discussion

NSConnection calls this method at the appropriate times. This method should not be called directly. This method could raise an NSInvalidSendPortException, NSInvalidReceivePortException, or an NSPortSendException, depending on the type of send port and the type of error.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSPort.h

sendBeforeDate:msgid:components:from:reserved:

This method is provided for subclasses that have custom types of `NSPort`.

```
- (BOOL)sendBeforeDate:(NSDate *)limitDate msgid:(NSUInteger)msgID
    components:(NSMutableArray *)components from:(NSPort *)receivePort
    reserved:(NSUInteger)headerSpaceReserved
```

Parameters

limitDate
The last instant that a message may be sent.

msgID
The message ID.

components
The message components.

receivePort
The receive port.

headerSpaceReserved
The number of bytes reserved for the header.

Discussion

`NSConnection` calls this method at the appropriate times. This method should not be called directly. This method could raise an `NSInvalidSendPortException`, `NSInvalidReceivePortException`, or an `NSPortSendException`, depending on the type of send port and the type of error.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

setDelegate:

Sets the receiver's delegate to a given object.

```
- (void)setDelegate:(id)anObject
```

Parameters

anObject
The delegate for the receiver.

Discussion

Does not retain *anObject*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- delegate` (page 853)

Declared In

`NSPort.h`

Delegate Methods

handlePortMessage:

Processes a given incoming message on the port.

```
- (void)handlePortMessage:(NSPortMessage *)portMessage
```

Parameters

portMessage

An incoming port message.

Discussion

See the `NSPortMessage` class specification for more information.

The delegate should implement only one of `handleMachMessage:` (page 556) and `handlePortMessage:`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

Notifications

NSPortDidBecomeInvalidNotification

Posted from the `invalidate` (page 853) method, which is invoked when the `NSPort` is deallocated or when it notices that its communication channel has been damaged. The notification object is the `NSPort` object that has become invalid. This notification does not contain a *`userInfo`* dictionary.

An `NSSocketPort` object cannot detect when its connection to a remote port is lost, even if the remote port is on the same machine. Therefore, it cannot invalidate itself and post this notification. Instead, you must detect the timeout error when the next message is sent.

The `NSPort` object posting this notification is no longer useful, so all receivers should unregister themselves for any notifications involving the `NSPort`. A method receiving this notification should check to see which port became invalid before attempting to do anything. In particular, observers that receive all `NSPortDidBecomeInvalidNotification` messages should be aware that communication with the window server is handled through an `NSPort`. If this port becomes invalid, drawing operations will cause a fatal error.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPort.h`

NSProcessInfo Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSProcessinfo.h
Companion guide:	Interacting with the Operating System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSProcessInfo` class provides methods to access information about the current process. Each process has a single, shared `NSProcessInfo` object, known as **process information agent**.

The process information agent can return such information as the arguments, environment variables, host name, or process name. The `processInfo` (page 861) class method returns the shared agent for the current process—that is, the process whose object sent the message. For example, the following line returns the `NSProcessInfo` object, which then provides the name of the current process:

```
NSString *processName = [[NSProcessInfo processInfo] processName];
```

The `NSProcessInfo` class also includes the `operatingSystem` (page 863) method, which returns an `enum` constant identifying the operating system on which the process is executing.

`NSProcessInfo` objects attempt to interpret environment variables and command-line arguments in the user's default C string encoding if they cannot be converted to Unicode as UTF-8 strings. If neither conversion works, these values are ignored by the `NSProcessInfo` object.

Tasks

Getting the Process Information Agent

`+ processInfo` (page 861)
Returns the process information agent for the process.

Accessing Process Information

`- arguments` (page 862)
Returns the command-line arguments for the process.

`- environment` (page 862)
Returns the variable names and their values in the environment from which the process was launched.

`- processIdentifier` (page 864)
Returns the identifier of the process.

`- globallyUniqueString` (page 862)
Returns a global unique identifier for the process.

`- processName` (page 864)
Returns the name of the process.

`- setProcessName:` (page 865)
Sets the name of the process.

Getting Host Information

`- hostName` (page 863)
Returns the name of the host computer.

`- operatingSystem` (page 863)
Returns a constant to indicate the operating system on which the process is executing.

`- operatingSystemName` (page 863)
Returns a string containing the name of the operating system on which the process is executing.

`- operatingSystemVersionString` (page 863)
Returns a string containing the version of the operating system on which the process is executing.

Getting Computer Information

- `physicalMemory` (page 864)
  Provides the amount of physical memory on the computer.
- `processorCount` (page 865)
  Provides the number of processing cores available on the computer.
- `activeProcessorCount` (page 861)
  Provides the number of active processing cores available on the computer.

Class Methods

processInfo

Returns the process information agent for the process.

```
+ (NSProcessInfo *)processInfo
```

Return Value
Shared process information agent for the process.

Discussion
An `NSProcessInfo` (page 859) object is created the first time this method is invoked, and that same object is returned on each subsequent invocation.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSProcessInfo.h`

Instance Methods

activeProcessorCount

Provides the number of active processing cores available on the computer.

```
- (NSUInteger)activeProcessorCount
```

Return Value
Number of active processing cores.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `processorCount` (page 865)

Declared In
NSProcessInfo.h

arguments

Returns the command-line arguments for the process.

```
- (NSArray *)arguments
```

Return Value
Array of strings with the process's command-line arguments.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

environment

Returns the variable names and their values in the environment from which the process was launched.

```
- (NSDictionary *)environment
```

Return Value
Dictionary of environment-variable names (keys) and their values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

globallyUniqueString

Returns a global unique identifier for the process.

```
- (NSString *)globallyUniqueString
```

Return Value
Global ID for the process. The ID includes the host name, process ID, and a time stamp, which ensures that the ID is unique for the network.

Discussion
This method generates a new string each time it is invoked, so it also uses a counter to guarantee that strings created from the same process are unique.

Availability
Available in iPhone OS 2.0 and later.

See Also
- processName (page 864)

Declared In
NSProcessInfo.h

hostName

Returns the name of the host computer.

```
- (NSString *)hostName
```

Return Value
Host name of the computer.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

operatingSystem

Returns a constant to indicate the operating system on which the process is executing.

```
- (unsigned int)operatingSystem
```

Return Value
Operating system identifier. See "Constants" (page 866) for a list of possible values. In Mac OS X, it's `NSMACHOperatingSystem`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

operatingSystemName

Returns a string containing the name of the operating system on which the process is executing.

```
- (NSString *)operatingSystemName
```

Return Value
Operating system name. In Mac OS X, it's `@"NSMACHOperatingSystem"`

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

operatingSystemVersionString

Returns a string containing the version of the operating system on which the process is executing.

```
- (NSString *)operatingSystemVersionString
```

Return Value
Operating system version. This string is human readable, localized, and is appropriate for displaying to the user. This string is *not* appropriate for parsing.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

physicalMemory

Provides the amount of physical memory on the computer.

```
- (unsigned long long)physicalMemory
```

Return Value
Amount of physical memory in bytes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProcessInfo.h

processIdentifier

Returns the identifier of the process.

```
- (int)processIdentifier
```

Return Value
Process ID of the process.

Availability
Available in iPhone OS 2.0 and later.

See Also
- processName (page 864)

Declared In
NSProcessInfo.h

processName

Returns the name of the process.

```
- (NSString *)processName
```

Return Value
Name of the process.

Discussion
The process name is used to register application defaults and is used in error messages. It does not uniquely identify the process.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `processIdentifier` (page 864)
- `setProcessName:` (page 865)

Declared In
`NSProcessInfo.h`

processorCount
Provides the number of processing cores available on the computer.

```
- (NSUInteger)processorCount
```

Return Value
Number of processing cores.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `activeProcessorCount` (page 861)

Declared In
`NSProcessInfo.h`

setProcessName:
Sets the name of the process.

```
- (void)setProcessName:(NSString *)name
```

Parameters
name

New name for the process.

Discussion

> **Warning:** User defaults and other aspects of the environment might depend on the process name, so be very careful if you change it. Setting the process name in this manner is not thread safe.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `processName` (page 864)

Declared In
NSProcessInfo.h

Constants

NSProcessInfo—Operating Systems

The following constants are provided by the NSProcessInfo class as return values for operatingSystem (page 863).

```
enum {
   NSWindowsNTOperatingSystem = 1,
   NSWindows95OperatingSystem,
   NSSolarisOperatingSystem,
   NSHPUXOperatingSystem,
   NSMACHOperatingSystem,
   NSSunOSOperatingSystem,
   NSOSF1OperatingSystem
};
```

Constants

NSHPUXOperatingSystem
Indicates the HP UX operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

NSMACHOperatingSystem
Indicates the Mac OS X operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

NSOSF1OperatingSystem
Indicates the OSF/1 operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

NSSolarisOperatingSystem
Indicates the Solaris operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

NSSunOSOperatingSystem
Indicates the Sun OS operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

NSWindows95OperatingSystem
Indicates the Windows 95 operating system.
Available in iPhone OS 2.0 and later.
Declared in NSProcessInfo.h

`NSWindowsNTOperatingSystem`

Indicates the Windows NT operating system.

Available in iPhone OS 2.0 and later.

Declared in `NSProcessInfo.h`

Declared In

`NSProcessInfo.h`

NSPropertyListSerialization Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSPropertyList.h
Companion guides:	Archives and Serializations Programming Guide for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSPropertyListSerialization` class provides methods that convert property list objects to and from several serialized formats. Property list objects include `NSData`, `NSString`, `NSArray`, `NSDictionary`, `NSDate`, and `NSNumber` objects. These objects are toll-free bridged with their respective Core Foundation types (`CFData`, `CFString`, and so on). For more about toll-free bridging, see Interchangeable Data Types.

Property list serialization automatically takes account of endianness on different platforms—for example, you can correctly read on an Intel-based Macintosh a binary property list created on a PowerPC-based Macintosh.

Tasks

Serializing a Property List

`+ dataFromPropertyList:format:errorDescription:` (page 870)
Returns an `NSData` object containing a given property list in a specified format.

Deserializing a Property List

`+ propertyListFromData:mutabilityOption:format:errorDescription:` (page 871)
Returns a property list object corresponding to the representation in a given `NSData` object.

Validating a Property List

`+ propertyList:isValidForFormat:` (page 871)
Returns a Boolean value that indicates whether a given property list is valid for a given format.

Class Methods

dataFromPropertyList:format:errorDescription:

Returns an `NSData` object containing a given property list in a specified format.

```
+ (NSData *)dataFromPropertyList:(id)plist format:(NSPropertyListFormat)format
    errorDescription:(NSString **)errorString
```

Parameters

plist
A property list object. *plist* must be a kind of `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary` object. Container objects must also contain only these kinds of objects.

format
A property list format. Possible values for *format* are described in NSPropertyListFormat (page 873).

errorString
Upon return, if the conversion is successful, *errorString* is `nil`. If the conversion fails, upon return contains a string describing the nature of the error. If you receive a string, you must release it.

Return Value

An `NSData` object containing *plist* in the format specified by *format*.

Special Considerations

Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ propertyListFromData:mutabilityOption:format:errorDescription: (page 871)

Declared In
NSPropertyList.h

propertyList:isValidForFormat:
Returns a Boolean value that indicates whether a given property list is valid for a given format.

```
+ (BOOL)propertyList:(id)plist isValidForFormat:(NSPropertyListFormat)format
```

Parameters
plist

A property list object.

format

A property list format. Possible values for *format* are listed in NSPropertyListFormat (page 873).

Return Value
YES if *plist* is a valid property list in format *format*, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPropertyList.h

propertyListFromData:mutabilityOption:format:errorDescription:
Returns a property list object corresponding to the representation in a given NSData object.

```
+ (id)propertyListFromData:(NSData *)data
    mutabilityOption:(NSPropertyListMutabilityOptions)opt
    format:(NSPropertyListFormat *)format errorDescription:(NSString **)errorString
```

Parameters
data

A data object containing a serialized property list.

opt

Determines whether the property list's contents are created as mutable objects, where possible. Possible values are described in NSPropertyListMutabilityOptions (page 872).

format

If the property list is valid, upon return contains the format. *format* can be NULL, in which case the property list format is not returned. Possible values are described in NSPropertyListFormat (page 873).

errorString

Upon return, if the conversion is successful, *errorString* is `nil`. If the conversion fails, upon return contains a string describing the nature of the error. If you receive a string, you must release it.

Return Value

A property list object corresponding to the representation in *data*. If data is not in a supported format, returns `nil`.

Special Considerations

Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ dataFromPropertyList:format:errorDescription:` (page 870)

Declared In

`NSPropertyList.h`

Constants

NSPropertyListMutabilityOptions

These constants specify mutability options in property lists.

```
typedef enum {
    NSPropertyListImmutable = kCFPropertyListImmutable,
    NSPropertyListMutableContainers = kCFPropertyListMutableContainers,
    NSPropertyListMutableContainersAndLeaves =
kCFPropertyListMutableContainersAndLeaves
} NSPropertyListMutabilityOptions;
```

Constants

`NSPropertyListImmutable`

Causes the returned property list to contain immutable objects.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

`NSPropertyListMutableContainers`

Causes the returned property list to have mutable containers but immutable leaves.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

`NSPropertyListMutableContainersAndLeaves`

Causes the returned property list to have mutable containers and leaves.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPropertyList.h

NSPropertyListFormat

These constants are used to specify a property list serialization format.

```
typedef enum {
    NSPropertyListOpenStepFormat = kCFPropertyListOpenStepFormat,
    NSPropertyListXMLFormat_v1_0 = kCFPropertyListXMLFormat_v1_0,
    NSPropertyListBinaryFormat_v1_0 = kCFPropertyListBinaryFormat_v1_0
} NSPropertyListFormat;
```

Constants

`NSPropertyListOpenStepFormat`
Specifies the old-style ASCII property list format inherited from the OpenStep APIs.

Important: The `NSPropertyListOpenStepFormat` constant is not supported for writing. It can be used only for reading old-style property lists.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

`NSPropertyListXMLFormat_v1_0`
Specifies the XML property list format.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

`NSPropertyListBinaryFormat_v1_0`
Specifies the binary property list format.

Available in iPhone OS 2.0 and later.

Declared in `NSPropertyList.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPropertyList.h

NSProxy Class Reference

Inherits from:	none (NSProxy is a root class)
Conforms to:	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSProxy.h
Companion guide:	Distributed Objects

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSProxy` is an abstract superclass defining an API for objects that act as stand-ins for other objects or for objects that don't exist yet. Typically, a message to a proxy is forwarded to the real object or causes the proxy to load (or transform itself into) the real object. Subclasses of `NSProxy` can be used to implement transparent distributed messaging (for example, `NSDistantObject`) or for lazy instantiation of objects that are expensive to create.

`NSProxy` implements the basic methods required of a root class, including those defined in the `NSObject` protocol. However, as an abstract class it doesn't provide an initialization method, and it raises an exception upon receiving any message it doesn't respond to. A concrete subclass must therefore provide an initialization or creation method and override the `forwardInvocation:` (page 880) and `methodSignatureForSelector:` (page 880) methods to handle messages that it doesn't implement

itself. A subclass's implementation of `forwardInvocation:` (page 880) should do whatever is needed to process the invocation, such as forwarding the invocation over the network or loading the real object and passing it the invocation. `methodSignatureForSelector:` (page 880) is required to provide argument type information for a given message; a subclass's implementation should be able to determine the argument types for the messages it needs to forward and should construct an `NSMethodSignature` object accordingly. See the `NSDistantObject`, `NSInvocation`, and `NSMethodSignature` class specifications for more information.

Adopted Protocols

NSObject
- `autorelease` (page 1303)
- `class` (page 1304)
- `conformsToProtocol:` (page 1304)
- `description` (page 1305)
- `hash` (page 1305)
- `isEqual:` (page 1306)
- `isKindOfClass:` (page 1306)
- `isMemberOfClass:` (page 1307)
- `isProxy` (page 1308)
- `performSelector:` (page 1308)
- `performSelector:withObject:` (page 1309)
- `performSelector:withObject:withObject:` (page 1309)
- `release` (page 1310)
- `respondsToSelector:` (page 1311)
- `retain` (page 1312)
- `retainCount` (page 1312)
- `self` (page 1313)
- `superclass` (page 1313)
- `zone` (page 1314)

Tasks

Creating Instances

`+ alloc` (page 877)
Returns a new instance of the receiving class

`+ allocWithZone:` (page 878)
Returns a new instance of the receiving class

Deallocating Instances

- `dealloc` (page 879)
  Deallocates the memory occupied by the receiver.

Finalizing an Object

- `finalize` (page 879)
  The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Handling Unimplemented Methods

- `forwardInvocation:` (page 880)
  Passes a given invocation to the real object the proxy represents.
- `methodSignatureForSelector:` (page 880)
  Raises `NSInvalidArgumentException`. Override this method in your concrete subclass to return a proper `NSMethodSignature` object for the given selector and the class your proxy objects stand in for.

Introspecting a Proxy Class

+ `respondsToSelector:` (page 878)
  Returns a Boolean value that indicates whether the receiving class responds to a given selector.

Describing a Proxy Class or Object

+ `class` (page 878)
  Returns `self` (the class object).
- `description` (page 879)
  Returns an `NSString` object containing the real class name and the `id` of the receiver as a hexadecimal number.

Class Methods

alloc

Returns a new instance of the receiving class

```
+ (id)alloc
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProxy.h

allocWithZone:

Returns a new instance of the receiving class

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

Return Value
A new instance of the receiving class, as described in the NSObject class specification under the allocWithZone: (page 783) class method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProxy.h

class

Returns self (the class object).

```
+ (Class)class
```

Return Value
self. Because this is a class method, it returns the class object

Availability
Available in iPhone OS 2.0 and later.

See Also
class (page 785) (NSObject)
class (page 1304) (NSObject protocol)

Declared In
NSProxy.h

respondsToSelector:

Returns a Boolean value that indicates whether the receiving class responds to a given selector.

```
+ (BOOL)respondsToSelector:(SEL)aSelector
```

Parameters
aSelector
A selector.

Return Value
YES if the receiving class responds to *aSelector* messages, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProxy.h

Instance Methods

dealloc

Deallocates the memory occupied by the receiver.

```
- (void)dealloc
```

Discussion
This method behaves as described in the NSObject class specification under the dealloc (page 799) instance method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- finalize (page 879)

Declared In
NSProxy.h

description

Returns an NSString object containing the real class name and the id of the receiver as a hexadecimal number.

```
- (NSString *)description
```

Return Value
An NSString object containing the real class name and the id of the receiver as a hexadecimal number.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProxy.h

finalize

The garbage collector invokes this method on the receiver before disposing of the memory it uses.

```
- (void)finalize
```

Discussion
This method behaves as described in the NSObject class specification under the finalize (page 800) instance method. Note that a finalize method must be thread-safe.

Availability
Available in iPhone OS 2.0 and later.

See Also
- dealloc (page 879)

Declared In
NSProxy.h

forwardInvocation:

Passes a given invocation to the real object the proxy represents.

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Parameters
anInvocation
The invocation to forward.

Discussion
NSProxy's implementation merely raises NSInvalidArgumentException. Override this method in your subclass to handle *anInvocation* appropriately, at the very least by setting its return value.

For example, if your proxy merely forwards messages to an instance variable named *realObject*, it can implement forwardInvocation: like this:

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    [anInvocation setTarget:realObject];
    [anInvocation invoke];
    return;
}
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSProxy.h

methodSignatureForSelector:

Raises NSInvalidArgumentException. Override this method in your concrete subclass to return a proper NSMethodSignature object for the given selector and the class your proxy objects stand in for.

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Parameters
aSelector
The selector for which to return a method signature.

Return Value
Not applicable. The implementation provided by NSProxy raises an exception.

Discussion

Be sure to avoid an infinite loop when necessary by checking that *aSelector* isn't the selector for this method itself and by not sending any message that might invoke this method.

For example, if your proxy merely forwards messages to an instance variable named *realObject*, it can implement `methodSignatureForSelector:` like this:

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    return [realObject methodSignatureForSelector:aSelector];
}
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`methodSignatureForSelector:` (page 805) (`NSObject`)

Declared In

`NSProxy.h`

NSRecursiveLock Class Reference

Inherits from:	NSObject
Conforms to:	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLock.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSRecursiveLock` defines a lock that may be acquired multiple times by the same thread without causing a deadlock, a situation where a thread is permanently blocked waiting for itself to relinquish a lock. While the locking thread has one or more locks, all other threads are prevented from accessing the code protected by the lock.

Adopted Protocols

NSLocking
- lock (page 1298)
- unlock (page 1298)

Tasks

Acquiring a Lock

- lockBeforeDate: (page 884)
  Attempts to acquire a lock before a given date.
- tryLock (page 885)
  Attempts to acquire a lock, and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

- setName: (page 885)
  Assigns a name to the receiver
- name (page 885)
  Returns the name associated with the receiver.

Instance Methods

lockBeforeDate:

Attempts to acquire a lock before a given date.

```
- (BOOL)lockBeforeDate:(NSDate *)limit
```

Parameters

limit
The time before which the lock should be acquired.

Return Value

YES if the lock is acquired before *limit*, otherwise NO.

Discussion

The thread is blocked until the receiver acquires the lock or *limit* is reached.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSLock.h

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setName: (page 885)

Declared In
NSLock.h

setName:

Assigns a name to the receiver

```
- (void)setName:(NSString *)newName
```

Parameters
newName
The new name for the receiver. This method makes a copy of the specified string.

Discussion
You can use a name string to identify a lock within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- name (page 885)

Declared In
NSLock.h

tryLock

Attempts to acquire a lock, and immediately returns a Boolean value that indicates whether the attempt was successful.

```
- (BOOL)tryLock
```

Return Value
YES if successful, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSLock.h

NSRunLoop Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSRunLoop.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSRunLoop` class declares the programmatic interface to objects that manage input sources. An `NSRunLoop` object processes input for sources such as mouse and keyboard events from the window system, `NSPort` objects, and `NSConnection` objects. An `NSRunLoop` object also processes `NSTimer` events.

In general, your application does not need to either create or explicitly manage `NSRunLoop` objects. Each `NSThread` object, including the application's main thread, has an `NSRunLoop` object automatically created for it as needed. If you need to access the current thread's run loop, you do so with the class method `currentRunLoop` (page 889).

Note that from the perspective of `NSRunloop`, `NSTimer` objects are not "input"—they are a special type, and one of the things that means is that they do not cause the run loop to return when they fire.

Warning: The `NSRunLoop` class is generally not considered to be thread-safe and its methods should only be called within the context of the current thread. You should never try to call the methods of an `NSRunLoop` object running in a different thread, as doing so might cause unexpected results.

Tasks

Accessing Run Loops and Modes

`+ currentRunLoop` (page 889)
Returns the `NSRunLoop` object for the current thread.

`- currentMode` (page 892)
Returns the receiver's current input mode.

`- limitDateForMode:` (page 893)
Performs one pass through the run loop in the specified mode and returns the date at which the next timer is scheduled to fire.

`+ mainRunLoop` (page 889)
Returns the run loop of the main thread.

`- getCFRunLoop` (page 893)
Returns the receiver's underlying *CFRunLoop Reference* object.

Managing Timers

`- addTimer:forMode:` (page 891)
Registers a given timer with a given input mode.

Managing Ports

`- addPort:forMode:` (page 890)
Adds a port as an input source to the specified mode of the run loop.

`- removePort:forMode:` (page 895)
Removes a port from the specified input mode of the run loop.

Running a Loop

`- run` (page 895)
Puts the receiver into a permanent loop, during which time it processes data from all attached input sources.

- runMode:beforeDate: (page 896)
  Runs the loop once, blocking for input in the specified mode until a given date.
- runUntilDate: (page 896)
  Runs the loop until the specified date, during which time it processes data from all attached input sources.
- acceptInputForMode:beforeDate: (page 890)
  Runs the loop once or until the specified date, accepting input only for the specified mode.

Scheduling and Canceling Messages

- performSelector:target:argument:order:modes: (page 894)
  Schedules the sending of a message on the current run loop.
- cancelPerformSelector:target:argument: (page 891)
  Cancels the sending of a previously scheduled message.
- cancelPerformSelectorsWithTarget: (page 892)
  Cancels all outstanding ordered performs scheduled with a given target.

Class Methods

currentRunLoop

Returns the NSRunLoop object for the current thread.

```
+ (NSRunLoop *)currentRunLoop
```

Return Value
The NSRunLoop object for the current thread.

Discussion
If a run loop does not yet exist for the thread, one is created and returned.

Availability
Available in iPhone OS 2.0 and later.

See Also
- currentMode (page 892)

Declared In
NSRunLoop.h

mainRunLoop

Returns the run loop of the main thread.

```
+ (NSRunLoop *)mainRunLoop
```

Return Value
An object representing the main thread's run loop.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRunLoop.h

Instance Methods

acceptInputForMode:beforeDate:

Runs the loop once or until the specified date, accepting input only for the specified mode.

```
- (void)acceptInputForMode:(NSString *)mode beforeDate:(NSDate *)limitDate
```

Parameters

mode
The mode in which to run. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 897).

limitDate
The date up until which to run.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the run loop once, returning as soon as one input source processes a message or the specifed time elapses.

Note: A timer is not considered an input source and may fire multiple times while waiting for this method to return

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Availability
Available in iPhone OS 2.0 and later.

See Also
- runMode:beforeDate: (page 896)

Declared In
NSRunLoop.h

addPort:forMode:

Adds a port as an input source to the specified mode of the run loop.

```
- (void)addPort:(NSPort *)aPort forMode:(NSString *)mode
```

Parameters

aPort

The port to add to the receiver.

mode

The mode in which to add *aPort*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 897).

Discussion

This method schedules the port with the receiver. You can add a port to multiple input modes. When the receiver is running in the specified mode, it dispatches messages destined for that port to the port's designated handler routine.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removePort:forMode: (page 895)

Declared In

NSRunLoop.h

addTimer:forMode:

Registers a given timer with a given input mode.

```
- (void)addTimer:(NSTimer *)aTimer forMode:(NSString *)mode
```

Parameters

aTimer

The timer to register with the receiver.

mode

The mode in which to add *aTimer*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 897).

Discussion

You can add a timer to multiple input modes. While running in the designated mode, the receiver causes the timer to fire on or after its scheduled fire date. Upon firing, the timer invokes its associated handler routine, which is a selector on a designated object.

The receiver retains *aTimer*. To remove a timer from all run loop modes on which it is installed, send an invalidate (page 1070) message to the timer.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSRunLoop.h

cancelPerformSelector:target:argument:

Cancels the sending of a previously scheduled message.

```
- (void)cancelPerformSelector:(SEL)aSelector target:(id)target
    argument:(id)anArgument
```

Parameters

aSelector

The previously-specified selector.

target

The previously-specified target.

anArgument

The previously-specified argument.

Discussion

You can use this method to cancel a message previously scheduled using the `performSelector:target:argument:order:modes:` (page 894) method. The parameters identify the message you want to cancel and must match those originally specified when the selector was scheduled. This method removes the perform request from all modes of the run loop.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRunLoop.h`

cancelPerformSelectorsWithTarget:

Cancels all outstanding ordered performs scheduled with a given target.

```
- (void)cancelPerformSelectorsWithTarget:(id)target
```

Parameters

target

The previously-specified target.

Discussion

This method cancels the previously scheduled messages associated with the target, ignoring the selector and argument of the scheduled operation. This is in contrast to `cancelPerformSelector:target:argument:` (page 891), which requires you to match the selector and argument as well as the target. This method removes the perform requests for the object from all modes of the run loop.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRunLoop.h`

currentMode

Returns the receiver's current input mode.

```
- (NSString *)currentMode
```

Return Value
The receiver's current input mode. This method returns the current input mode *only* while the receiver is running; otherwise, it returns `nil`.

Discussion
The current mode is set by the methods that run the run loop, such as `acceptInputForMode:beforeDate:` (page 890) and `runMode:beforeDate:` (page 896).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ currentRunLoop` (page 889)
`- limitDateForMode:` (page 893)
`- run` (page 895)
`- runUntilDate:` (page 896)

Declared In
`NSRunLoop.h`

getCFRunLoop
Returns the receiver's underlying *CFRunLoop Reference* object.

```
- (CFRunLoopRef)getCFRunLoop
```

Return Value
The receiver's underlying *CFRunLoop Reference* object.

Discussion
You can use the returned run loop to configure the current run loop using Core Foundation function calls. For example, you might use this function to set up a run loop observer.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSRunLoop.h`

limitDateForMode:
Performs one pass through the run loop in the specified mode and returns the date at which the next timer is scheduled to fire.

```
- (NSDate *)limitDateForMode:(NSString *)mode
```

Parameters
mode

The run loop mode to search. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 897).

Return Value
The date at which the next timer is scheduled to fire, or `nil` if there are no input sources for this mode.

Discussion
The run loop is entered with an immediate timeout, so the run loop does not block, waiting for input, if no input sources need processing.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRunLoop.h

performSelector:target:argument:order:modes:
Schedules the sending of a message on the current run loop.

```
- (void)performSelector:(SEL)aSelector target:(id)target argument:(id)anArgument
    order:(NSUInteger)order modes:(NSArray *)modes
```

Parameters
aSelector
: A selector that identifies the method to invoke. This method should not have a significant return value and should take a single argument of type `id`.

target
: The object that defines the selector in *aSelector*.

anArgument
: The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

order
: The priority for the message. If multiple messages are scheduled, the messages with a lower order value are sent before messages with a higher order value.

modes
: An array of input modes for which the message may be sent. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 897).

Discussion
This method sets up a timer to perform the *aSelector* message on the current thread's run loop at the start of the next run loop iteration. The timer is configured to run in the modes specified by the *modes* parameter. When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in one of the specified modes; otherwise, the timer waits until the run loop is in one of those modes.

This method returns before the *aSelector* message is sent. The receiver retains the *target* and *anArgument* objects until the timer for the selector fires, and then releases them as part of its cleanup.

Use this method if you want multiple messages to be sent after the current event has been processed and you want to make sure these messages are sent in a certain order.

Availability
Available in iPhone OS 2.0 and later.

See Also
- cancelPerformSelector:target:argument: (page 891)

Declared In
NSRunLoop.h

removePort:forMode:

Removes a port from the specified input mode of the run loop.

```
- (void)removePort:(NSPort *)aPort forMode:(NSString *)mode
```

Parameters

aPort

The port to remove from the receiver.

mode

The mode from which to remove *aPort*. You may specify a custom mode or use one of the modes listed in "Run Loop Modes" (page 897).

Discussion
If you added the port to multiple input modes, you must remove it from each mode separately.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addPort:forMode: (page 890)

Declared In
NSRunLoop.h

run

Puts the receiver into a permanent loop, during which time it processes data from all attached input sources.

```
- (void)run
```

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the receiver in the NSDefaultRunLoopMode by repeatedly invoking runMode:beforeDate: (page 896). In other words, this method effectively begins an infinite loop that processes data from the run loop's input sources and timers.

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

If you want the run loop to terminate, you shouldn't use this method. Instead, use one of the other run methods and also check other arbitrary conditions of your own, in a loop. A simple example would be:

```
BOOL shouldKeepRunning = YES;        // global
NSRunLoop *theRL = [NSRunLoop currentRunLoop];
while (shouldKeepRunning && [theRL runMode:NSDefaultRunLoopMode beforeDate:[NSDate
 distantFuture]]);
```

where `shouldKeepRunning` is set to `NO` somewhere else in the program.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- runUntilDate:` (page 896)

Declared In
`NSRunLoop.h`

runMode:beforeDate:

Runs the loop once, blocking for input in the specified mode until a given date.

```
- (BOOL)runMode:(NSString *)mode beforeDate:(NSDate *)limitDate
```

Parameters

mode
The mode in which to run. You may specify custom modes or use one of the modes listed in "Run Loop Modes" (page 897).

limitDate
The date until which to block.

Return Value
`NO` without starting the run loop if there are no input sources in *mode*; otherwise `YES`.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it returns after either the first input source is processed or *limitDate* is reached. Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X may install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Note: A timer is not considered an input source and may fire multiple times while waiting for this method to return

Availability
Available in iPhone OS 2.0 and later.

See Also
`- run` (page 895)
`- runUntilDate:` (page 896)

Declared In
`NSRunLoop.h`

runUntilDate:

Runs the loop until the specified date, during which time it processes data from all attached input sources.

```
- (void)runUntilDate:(NSDate *)limitDate
```

Parameters

limitDate

The date up until which to run.

Discussion

If no input sources or timers are attached to the run loop, this method exits immediately; otherwise, it runs the receiver in the `NSDefaultRunLoopMode` by repeatedly invoking `runMode:beforeDate:` (page 896) until the specified expiration date.

Manually removing all known input sources and timers from the run loop is not a guarantee that the run loop will exit. Mac OS X can install and remove additional input sources as needed to process requests targeted at the receiver's thread. Those sources could therefore prevent the run loop from exiting.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- run` (page 895)

Declared In

`NSRunLoop.h`

Constants

Run Loop Modes

`NSRunLoop` defines the following run loop mode.

```
extern NSString *NSDefaultRunLoopMode;
```

Constants

`NSDefaultRunLoopMode`

The mode to deal with input sources other than `NSConnection` objects.

This is the most commonly used run-loop mode.

Available in iPhone OS 2.0 and later.

Declared in `NSRunLoop.h`

`NSRunLoopCommonModes`

Objects added to a run loop using this value as the mode are monitored by all run loop modes that have been declared as a member of the set of "common" modes; see the description of `CFRunLoopAddCommonMode` for details.

Available in Mac OS X v10.5 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSRunLoop.h`

Declared In

`Foundation/NSRunLoop.h`

Additional run loop modes are defined by `NSConnection` and `NSApplication`.

`NSConnectionReplyMode`	Use this mode to indicate NSConnection objects waiting for replies. Defined in the Foundation/NSConnection.h header file. You rarely need to use this mode.
`NSModalPanelRunLoopMode`	A run loop should be set to this mode when waiting for input from a modal panel, such as NSSavePanel or NSOpenPanel.
`NSEventTrackingRunLoopMode`	A run loop should be set to this mode when tracking events modally, such as a mouse-dragging loop.

NSScanner Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSScanner.h Foundation/NSDecimalNumber.h
Companion guide:	String Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSScanner` class is an abstract superclass of a class cluster that declares the programmatic interface for an object that scans values from an `NSString` object.

An `NSScanner` object interprets and converts the characters of an `NSString` object into number and string values. You assign the scanner's string on creating it, and the scanner progresses through the characters of that string from beginning to end as you request items.

Because of the nature of class clusters, scanner objects aren't actual instances of the `NSScanner` class but one of its private subclasses. Although a scanner object's class is private, its interface is public, as declared by this abstract superclass, `NSScanner`. The primitive methods of `NSScanner` are `string` (page 913) and all of the methods listed under "Configuring a Scanner" (page 900) in the "Methods by Task" section. The objects you create using this class are referred to as scanner objects (and when no confusion will result, merely as scanners).

You can set an `NSScanner` object to ignore a set of characters as it scans the string using the `setCharactersToBeSkipped:` (page 912) method. The default set of characters to skip is the whitespace and newline character set.

To retrieve the unscanned remainder of the string, use `[[scanner string]substringFromIndex:` (page 1038)`[scanner scanLocation]]`.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 1250)

Tasks

Creating an Scanner

`+ scannerWithString:` (page 902)
Returns an `NSScanner` object that scans a given string.

`+ localizedScannerWithString:` (page 902)
Returns an `NSScanner` object that scans a given string according to the user's default locale.

`- initWithString:` (page 904)
Returns an `NSScanner` object initialized to scan a given string.

Getting a Scanner's String

`- string` (page 913)
Returns the string with which the receiver was created or initialized.

Configuring a Scanner

`- setScanLocation:` (page 913)
Sets the location at which the next scan operation will begin to a given index.

`- scanLocation` (page 909)
Returns the character position at which the receiver will begin its next scanning operation.

`- setCaseSensitive:` (page 911)
Sets whether the receiver is case sensitive when scanning characters.

- `caseSensitive` (page 903)
  Returns a Boolean value that indicates whether the receiver distinguishes case in the characters it scans.
- `setCharactersToBeSkipped:` (page 912)
  Sets the set of characters to ignore when scanning for a value representation.
- `charactersToBeSkipped` (page 903)
  Returns a character set containing the characters the receiver ignores when looking for a scannable element.
- `setLocale:` (page 912)
  Sets the receiver's locale to a given locale.
- `locale` (page 904)
  Returns the receiver's locale.

Scanning a String

- `scanCharactersFromSet:intoString:` (page 905)
  Scans the string as long as characters from a given character set are encountered, accumulating characters into a string that's returned by reference.
- `scanUpToCharactersFromSet:intoString:` (page 910)
  Scans the string until a character from a given character set is encountered, accumulating characters into a string that's returned by reference.
- `scanDecimal:` (page 905)
  Scans for an `NSDecimal` value, returning a found value by reference.
- `scanDouble:` (page 906)
  Scans for a `double` value, returning a found value by reference.
- `scanFloat:` (page 906)
  Scans for a `float` value, returning a found value by reference.
- `scanInteger:` (page 908)
  Scans for an `NSInteger` value from a decimal representation, returning a found value by reference
- `scanInt:` (page 907)
  Scans for an `int` value from a decimal representation, returning a found value by reference.
- `scanHexInt:` (page 907)
  Scans for an `unsigned` value from a hexadecimal representation, returning a found value by reference.
- `scanLongLong:` (page 909)
  Scans for a `long long` value from a decimal representation, returning a found value by reference.
- `scanString:intoString:` (page 909)
  Scans a given string, returning an equivalent string object by reference if a match is found.
- `scanUpToString:intoString:` (page 911)
  Scans the string until a given string is encountered, accumulating characters into a string that's returned by reference.

- isAtEnd (page 904)
  Returns a Boolean value that indicates whether the receiver has exhausted all significant characters

Class Methods

localizedScannerWithString:

Returns an NSScanner object that scans a given string according to the user's default locale.

```
+ (id)localizedScannerWithString:(NSString *)aString
```

Parameters

aString
The string to scan.

Return Value

An NSScanner object that scans *aString* according to the user's default locale.

Discussion

Sets the string to scan by invoking initWithString: (page 904) with *aString*. The locale is set with setLocale: (page 912).

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSScanner.h

scannerWithString:

Returns an NSScanner object that scans a given string.

```
+ (id)scannerWithString:(NSString *)aString
```

Parameters

aString
The string to scan.

Return Value

An NSScanner object that scans *aString*.

Discussion

Sets the string to scan by invoking initWithString: (page 904) with *aString*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSScanner.h

Instance Methods

caseSensitive

Returns a Boolean value that indicates whether the receiver distinguishes case in the characters it scans.

```
- (BOOL)caseSensitive
```

Return Value

`YES` if the receiver distinguishes case in the characters it scans, otherwise `NO`.

Discussion

Scanners are not case sensitive by default. Note that case sensitivity doesn't apply to the characters to be skipped.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setCaseSensitive:` (page 911)
- `setCharactersToBeSkipped:` (page 912)

Declared In

`NSScanner.h`

charactersToBeSkipped

Returns a character set containing the characters the receiver ignores when looking for a scannable element.

```
- (NSCharacterSet *)charactersToBeSkipped
```

Return Value

A character set containing the characters the receiver ignores when looking for a scannable element.

Discussion

For example, if a scanner ignores spaces and you send it a `scanInt:` (page 907) message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using `scanInt:` (page 907) when the set of characters to be skipped is the decimal digits), the result is undefined.

The default set to skip is the whitespace and newline character set.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setCharactersToBeSkipped:` (page 912)

`whitespaceAndNewlineCharacterSet` (page 142) (`NSCharacterSet`)

Declared In
NSScanner.h

initWithString:

Returns an NSScanner object initialized to scan a given string.

```
- (id)initWithString:(NSString *)aString
```

Parameters
aString
The string to scan.

Return Value
An NSScanner object initialized to scan *aString* from the beginning. The returned object might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ localizedScannerWithString: (page 902)
+ scannerWithString: (page 902)

Declared In
NSScanner.h

isAtEnd

Returns a Boolean value that indicates whether the receiver has exhausted all significant characters

```
- (BOOL)isAtEnd
```

Return Value
YES if the receiver has exhausted all significant characters in its string, otherwise NO.

If only characters from the set to be skipped remain, returns YES.

Availability
Available in iPhone OS 2.0 and later.

See Also
- charactersToBeSkipped (page 903)

Declared In
NSScanner.h

locale

Returns the receiver's locale.

```
- (id)locale
```

Return Value
The receiver's locale, or `nil` if it has none.

Discussion
A scanner's locale affects the way it interprets numeric values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A scanner with no locale set uses non-localized values.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setLocale:` (page 912)

Declared In
`NSScanner.h`

scanCharactersFromSet:intoString:

Scans the string as long as characters from a given character set are encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanCharactersFromSet:(NSCharacterSet *)scanSet intoString:(NSString
    **)stringValue
```

Parameters

scanSet
The set of characters to scan.

stringValue
Upon return, contains the characters scanned.

Return Value
`YES` if the receiver scanned any characters, otherwise `NO`.

Discussion
Invoke this method with `NULL` as *stringValue* to simply scan past a given set of characters.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- scanUpToCharactersFromSet:intoString:` (page 910)

Declared In
`NSScanner.h`

scanDecimal:

Scans for an `NSDecimal` value, returning a found value by reference.

```
- (BOOL)scanDecimal:(NSDecimal *)decimalValue
```

Parameters

decimalValue

Upon return, contains the scanned value. See the `NSDecimalNumber` class specification for more information about `NSDecimal` values.

Return Value

`YES` if the receiver finds a valid `NSDecimal` representation, otherwise `NO`.

Discussion

Invoke this method with `NULL` as *decimalValue* to simply scan past an `NSDecimal` representation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimalNumber.h`

scanDouble:

Scans for a `double` value, returning a found value by reference.

```
- (BOOL)scanDouble:(double *)doubleValue
```

Parameters

doubleValue

Upon return, contains the scanned value. Contains `HUGE_VAL` or `-HUGE_VAL` on overflow, or `0.0` on underflow.

Return Value

`YES` if the receiver finds a valid floating-point representation, otherwise `NO`.

Discussion

Skips past excess digits in the case of overflow, so the scanner's position is past the entire floating-point representation.

Invoke this method with `NULL` as *doubleValue* to simply scan past a `double` value representation. Floating-point representations are assumed to be IEEE compliant.

Availability

Available in iPhone OS 2.0 and later.

See Also

`doubleValue` (page 989) (`NSString`)

Declared In

`NSScanner.h`

scanFloat:

Scans for a `float` value, returning a found value by reference.

```
- (BOOL)scanFloat:(float *)floatValue
```

Parameters

floatValue

Upon return, contains the scanned value. Contains HUGE_VAL or -HUGE_VAL on overflow, or 0.0 on underflow.

Return Value

YES if the receiver finds a valid floating-point representation, otherwise NO.

Discussion

Skips past excess digits in the case of overflow, so the scanner's position is past the entire floating-point representation.

Invoke this method with NULL as *floatValue* to simply scan past a float value representation. Floating-point representations are assumed to be IEEE compliant.

Availability

Available in iPhone OS 2.0 and later.

See Also

floatValue (page 990) (NSString)

Declared In

NSScanner.h

scanHexInt:

Scans for an unsigned value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexInt:(unsigned *)intValue
```

Parameters

intValue

Upon return, contains the scanned value. Contains INT_MAX or INT_MIN on overflow.

Return Value

Returns YES if the receiver finds a valid hexadecimal integer representation, otherwise NO.

Discussion

The hexadecimal integer representation may optionally be preceded by 0x or 0X. Skips past excess digits in the case of overflow, so the receiver's position is past the entire hexadecimal representation.

Invoke this method with NULL as *intValue* to simply scan past a hexadecimal integer representation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSScanner.h

scanInt:

Scans for an int value from a decimal representation, returning a found value by reference.

```
- (BOOL)scanInt:(int *)intValue
```

Parameters

intValue

Upon return, contains the scanned value. Contains INT_MAX or INT_MIN on overflow.

Return Value

YES if the receiver finds a valid decimal integer representation, otherwise NO.

Discussion

Skips past excess digits in the case of overflow, so the receiver's position is past the entire decimal representation.

Invoke this method with NULL as *intValue* to simply scan past a decimal integer representation.

Availability

Available in iPhone OS 2.0 and later.

See Also

intValue (page 1009) (NSString)

- scanInteger: (page 908)

Declared In

NSScanner.h

scanInteger:

Scans for an NSInteger value from a decimal representation, returning a found value by reference

```
- (BOOL)scanInteger:(NSInteger *)value
```

Parameters

value

Upon return, contains the scanned value.

Return Value

YES if the receiver finds a valid integer representation, otherwise NO.

Discussion

Skips past excess digits in the case of overflow, so the receiver's position is past the entire integer representation.

Invoke this method with NULL as *value* to simply scan past a decimal integer representation.

Availability

Available in iPhone OS 2.0 and later.

See Also

integerValue (page 1008) (NSString)

- scanInt: (page 907)

Declared In

NSScanner.h

scanLocation

Returns the character position at which the receiver will begin its next scanning operation.

```
- (NSUInteger)scanLocation
```

Return Value
The character position at which the receiver will begin its next scanning operation.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setScanLocation: (page 913)

Declared In
NSScanner.h

scanLongLong:

Scans for a long long value from a decimal representation, returning a found value by reference.

```
- (BOOL)scanLongLong:(long long *)longLongValue
```

Parameters
longLongValue

Upon return, contains the scanned value. Contains LONG_LONG_MAX or LONG_LONG_MIN on overflow.

Return Value
YES if the receiver finds a valid decimal integer representation, otherwise NO.

Discussion
All overflow digits are skipped. Skips past excess digits in the case of overflow, so the receiver's position is past the entire decimal representation.

Invoke this method with NULL as *longLongValue* to simply scan past a long decimal integer representation.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSScanner.h

scanString:intoString:

Scans a given string, returning an equivalent string object by reference if a match is found.

```
- (BOOL)scanString:(NSString *)string intoString:(NSString **)stringValue
```

Parameters
string

The string for which to scan at the current scan location.

stringValue
Upon return, if the receiver contains a string equivalent to *string* at the current scan location, contains a string equivalent to *string*.

Return Value
YES if *stringValue* matches the characters at the scan location, otherwise NO.

Discussion
If *string* is present at the current scan location, then the current scan location is advanced to after the string; otherwise the scan location does not change.

Invoke this method with NULL as *stringValue* to simply scan past a given string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- scanUpToString:intoString: (page 911)

Declared In
NSScanner.h

scanUpToCharactersFromSet:intoString:

Scans the string until a character from a given character set is encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanUpToCharactersFromSet:(NSCharacterSet *)stopSet intoString:(NSString
    **)stringValue
```

Parameters
stopSet
The set of characters up to which to scan.

stringValue
Upon return, contains the characters scanned.

Return Value
YES if the receiver scanned any characters, otherwise NO.

If the only scanned characters are in the charactersToBeSkipped (page 903) character set (which is the whitespace and newline character set by default), then returns NO.

Discussion
Invoke this method with NULL as *stringValue* to simply scan up to a given set of characters.

If no characters in *stopSet* are present in the scanner's source string, the remainder of the source string is put into *stringValue*, the receiver's scanLocation is advanced to the end of the source string, and the method returns YES.

Availability
Available in iPhone OS 2.0 and later.

See Also
- scanCharactersFromSet:intoString: (page 905)

Declared In
NSScanner.h

scanUpToString:intoString:

Scans the string until a given string is encountered, accumulating characters into a string that's returned by reference.

```
- (BOOL)scanUpToString:(NSString *)stopString intoString:(NSString **)stringValue
```

Parameters

stopString
The string to scan up to.

stringValue
Upon return, contains any characters that were scanned.

Return Value
YES if the receiver scans any characters, otherwise NO.

If the only scanned characters are in the charactersToBeSkipped (page 903) character set (which by default is the whitespace and newline character set), then this method returns NO.

Discussion
If stopString is present in the receiver, then on return the scan location is set to the beginning of that string.

If the search string (stopString) isn't present in the scanner's source string, the remainder of the source string is put into *stringValue*, the receiver's scanLocation is advanced to the end of the source string, and the method returns YES.

Invoke this method with NULL as *stringValue* to simply scan up to a given string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- scanString:intoString: (page 909)

Declared In
NSScanner.h

setCaseSensitive:

Sets whether the receiver is case sensitive when scanning characters.

```
- (void)setCaseSensitive:(BOOL)flag
```

Parameters

flag
If YES, the receiver will distinguish case when scanning characters, otherwise it will ignore case distinctions.

Discussion
Scanners are not case sensitive by default. Note that case sensitivity doesn't apply to the characters to be skipped.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `caseSensitive` (page 903)
- `setCharactersToBeSkipped:` (page 912)

Declared In
`NSScanner.h`

setCharactersToBeSkipped:
Sets the set of characters to ignore when scanning for a value representation.

```
- (void)setCharactersToBeSkipped:(NSCharacterSet *)skipSet
```

Parameters
skipSet
The characters to ignore when scanning for a value representation.

Discussion
For example, if a scanner ignores spaces and you send it a `scanInt:` (page 907) message, it skips spaces until it finds a decimal digit or other character. While an element is being scanned, however, no characters are skipped. If you scan for something made of characters in the set to be skipped (for example, using `scanInt:` (page 907) when the set of characters to be skipped is the decimal digits), the result is undefined.

The characters to be skipped are treated literally as single values. A scanner doesn't apply its case sensitivity setting to these characters and doesn't attempt to match composed character sequences with anything in the set of characters to be skipped (though it does match pre-composed characters individually). If you want to skip all vowels while scanning a string, for example, you can set the characters to be skipped to those in the string "AEIOUaeiou" (plus any accented variants with pre-composed characters).

The default set of characters to skip is the whitespace and newline character set.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `charactersToBeSkipped` (page 903)
- `whitespaceAndNewlineCharacterSet` (page 142) (`NSCharacterSet`)

Declared In
`NSScanner.h`

setLocale:
Sets the receiver's locale to a given locale.

```
- (void)setLocale:(id)aLocale
```

Parameters

aLocale

The locale for the receiver.

Discussion

A scanner's locale affects the way it interprets values from the string. In particular, a scanner uses the locale's decimal separator to distinguish the integer and fractional parts of floating-point representations. A new scanner's locale is by default `nil`, which causes it to use non-localized values.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `locale` (page 904)

Declared In

`NSScanner.h`

setScanLocation:

Sets the location at which the next scan operation will begin to a given index.

```
- (void)setScanLocation:(NSUInteger)index
```

Parameters

index

The location at which the next scan operation will begin. Raises an `NSRangeException` if *index* is beyond the end of the string being scanned.

Discussion

This method is useful for backing up to rescan after an error.

Rather than setting the scan location directly to skip known sequences of characters, use `scanString:intoString:` (page 909) or `scanCharactersFromSet:intoString:` (page 905), which allow you to verify that the expected substring (or set of characters) is in fact present.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `scanLocation` (page 909)

Declared In

`NSScanner.h`

string

Returns the string with which the receiver was created or initialized.

```
- (NSString *)string
```

Return Value
The string with which the receiver was created or initialized.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `locale` (page 904)

Declared In
`NSScanner.h`

NSSet Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSSet.h
Companion guide:	Collections Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSSet`, `NSMutableSet`, and `NSCountedSet` classes declare the programmatic interface to an object that manages a set of objects. `NSSet` provides support for the mathematical concept of a set. A set, both in its mathematical sense and in the implementation of `NSSet`, is an unordered collection of distinct elements. The `NSMutableSet` (a subclass of `NSSet`) and `NSCountedSet` (a subclass of `NSMutableSet`) classes are provided for sets whose contents may be altered.

NSSet and NSMutableSet are part of a class cluster, so sets are not actual instances of NSSet or NSMutableSet. Rather, the instances belong to one of their private subclasses. (For convenience, we use the term set to refer to any one of these instances without specifying its exact class membership.) Although a set's class is private, its interface is public, as declared by the abstract superclasses NSSet and NSMutableSet. Note that NSCountedSet is not part of the class cluster; it is a concrete subclass of NSMutableSet.

NSSet declares the programmatic interface for static sets of objects. You establish a static set's entries when it's created, and thereafter the entries can't be modified. NSMutableSet, on the other hand, declares a programmatic interface for dynamic sets of objects. A dynamic—or mutable—set allows the addition and deletion of entries at any time, automatically allocating memory as needed.

You can use sets as an alternative to arrays when the order of elements isn't important and performance in testing whether an object is contained in the set is a consideration—while arrays are ordered, testing for membership is slower than with sets.

Objects in a set must respond to the NSObject protocol methods hash (page 1305) and isEqual: (page 1306)—see the NSObject protocol for more information.

Note that if mutable objects are stored in a set, either the hash method of the objects shouldn't depend on the internal state of the mutable objects or the mutable objects shouldn't be modified while they're in the set (note that it can be difficult to know whether or not a given object is in a collection).

Objects added to a set are not copied; rather, an object receives a retain message before it's added to a set.

Typically, you create a temporary set by sending one of the set... methods to the NSSet class object. These methods return an NSSet object containing the elements (if any) you pass in as arguments. The set (page 919) method is a "convenience" method to create an empty mutable set.

The set classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert a set of one type to the other.

NSSet provides methods for querying the elements of the set. allObjects (page 923) returns an array containing the objects in a set. anyObject (page 924) returns some object in the set. count (page 925) returns the number of objects currently in the set. member: (page 931) returns the object in the set that is equal to a specified object. Additionally, intersectsSet: (page 929) tests for set intersection, isEqualToSet: (page 929) tests for set equality, and isSubsetOfSet: (page 930) tests for one set being a subset of another.

The objectEnumerator (page 931) method provides for traversing elements of the set one by one. For better performance on Mac OS X v10.5 and later, you can also use the Objective-C fast enumeration feature (see Fast Enumeration).

NSSet's makeObjectsPerformSelector: (page 930) and makeObjectsPerformSelector:withObject: (page 930) methods provides for sending messages to individual objects in the set.

NSSet is "toll-free bridged" with its Core Foundation counterpart, *CFSet Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an NSSet * parameter, you can pass a CFSetRef, and in a function where you see a CFSetRef parameter, you can pass an NSSet instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- copyWithZone: (page 1250)

NSMutableCopying
- mutableCopyWithZone: (page 1300)

Tasks

Creating a Set

+ set (page 919)
Creates and returns an empty set.

+ setWithArray: (page 920)
Creates and returns a set containing a uniqued collection of those objects contained in a given array.

+ setWithObject: (page 920)
Creates and returns a set that contains a single given object.

+ setWithObjects: (page 921)
Creates and returns a set containing the objects in a given argument list.

+ setWithObjects:count: (page 921)
Creates and returns a set containing a specified number of objects from a given C array of objects.

+ setWithSet: (page 922)
Creates and returns a set containing the objects from another set.

- setByAddingObject: (page 933)
Returns a new set formed by adding a given object to the collection defined by the receiver.

- setByAddingObjectsFromSet: (page 934)
Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

- setByAddingObjectsFromArray: (page 933)
Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

Initializing a Set

- initWithArray: (page 926)
Initializes a newly allocated set with the objects that are contained in a given array.

- `initWithObjects:` (page 926)
  Initializes a newly allocated set with members taken from the specified list of objects.
- `initWithObjects:count:` (page 927)
  Initializes a newly allocated set with a specified number of objects from a given C array of objects.
- `initWithSet:` (page 927)
  Initializes a newly allocated set and adds to it objects from another given set.
- `initWithSet:copyItems:` (page 928)
  Initializes a newly allocated set and adds to it members of another given set.

Counting Entries

- `count` (page 925)
  Returns the number of members in the receiver.

Accessing Set Members

- `allObjects` (page 923)
  Returns an array containing the receiver's members, or an empty array if the receiver has no members.
- `anyObject` (page 924)
  Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.
- `containsObject:` (page 924)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `makeObjectsPerformSelector:` (page 930)
  Sends to each object in the receiver a message specified by a given selector.
- `makeObjectsPerformSelector:withObject:` (page 930)
  Sends to each object in the receiver a message specified by a given selector.
- `member:` (page 931)
  Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.
- `objectEnumerator` (page 931)
  Returns an enumerator object that lets you access each object in the receiver.

Comparing Sets

- `isSubsetOfSet:` (page 930)
  Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.
- `intersectsSet:` (page 929)
  Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.
- `isEqualToSet:` (page 929)
  Compares the receiver to another set.

- valueForKey: (page 934)
  Return a set containing the results of invoking valueForKey: on each of the receiver's members.
- setValue:forKey: (page 934)
  Invokes setValue:forKey: on each of the receiver's members.

Key-Value Observing

- addObserver:forKeyPath:options:context: (page 923)
  Raises an exception.
- removeObserver:forKeyPath: (page 932)
  Raises an exception.

Describing a Set

- description (page 925)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- descriptionWithLocale: (page 925)
  Returns a string that represents the contents of the receiver, formatted as a property list.

Class Methods

set

Creates and returns an empty set.

```
+ (id)set
```

Return Value
A new empty set.

Discussion
This method is declared primarily for the use of mutable subclasses of NSSet.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ setWithArray: (page 920)
+ setWithObject: (page 920)
+ setWithObjects: (page 921)
- setByAddingObject: (page 933)
- setByAddingObjectsFromSet: (page 934)
- setByAddingObjectsFromArray: (page 933)

Declared In
NSSet.h

setWithArray:

Creates and returns a set containing a uniqued collection of those objects contained in a given array.

```
+ (id)setWithArray:(NSArray *)anArray
```

Parameters

anArray

An array containing the objects to add to the new set. If the same object appears more than once *objects*, it is added only once to the returned set. Each object receives a `retain` (page 1312) message as it is added to the set.

Return Value

A new set containing a uniqued collection of those objects contained in *anArray*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ set` (page 919)
`+ setWithObject:` (page 920)
`+ setWithObjects:` (page 921)
`- setByAddingObject:` (page 933)
`- setByAddingObjectsFromSet:` (page 934)
`- setByAddingObjectsFromArray:` (page 933)

Declared In

`NSSet.h`

setWithObject:

Creates and returns a set that contains a single given object.

```
+ (id)setWithObject:(id)anObject
```

Parameters

anObject

The object to add to the new set. *anObject* receives a `retain` (page 1312) message after being added to the set.

Return Value

A new set that contains a single member, *anObject*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ set` (page 919)
`+ setWithArray:` (page 920)
`+ setWithObjects:` (page 921)
`- setByAddingObject:` (page 933)
`- setByAddingObjectsFromSet:` (page 934)
`- setByAddingObjectsFromArray:` (page 933)

Declared In
NSSet.h

setWithObjects:

Creates and returns a set containing the objects in a given argument list.

```
+ (id)setWithObjects:(id)anObject ...
```

Parameters

anObject
The first object to add to the new set.

...
A comma-separated list of objects, ending with `nil`, to add to the new set. If the same object appears more than once *objects*, it is added only once to the returned set. Each object receives a `retain` (page 1312) message as it is added to the set.

Return Value
A new set containing the objects in the argument list.

Discussion
As an example, the following code excerpt creates a set containing three different types of elements (assuming `aPath` exits):

```
NSSet *mySet;
NSData *someData = [NSData dataWithContentsOfFile:aPath];
NSValue *aValue = [NSNumber numberWithInt:5];
NSString *aString = @"a string";

mySet = [NSSet setWithObjects:someData, aValue, aString, nil];
```

Availability
Available in iPhone OS 2.0 and later.

See Also
+ `set` (page 919)
+ `setWithArray:` (page 920)
+ `setWithObject:` (page 920)
- `setByAddingObject:` (page 933)
- `setByAddingObjectsFromSet:` (page 934)
- `setByAddingObjectsFromArray:` (page 933)

Declared In
NSSet.h

setWithObjects:count:

Creates and returns a set containing a specified number of objects from a given C array of objects.

```
+ (id)setWithObjects:(id *)objects count:(NSUInteger)count
```

Parameters

objects

A C array of objects to add to the new set. If the same object appears more than once *objects*, it is added only once to the returned set. Each object receives a `retain` (page 1312) message as it is added to the set.

count

The number of objects from *objects* to add to the new set.

Return Value

A new set containing *count* objects from the list of objects specified by *objects*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ set` (page 919)
`+ setWithArray:` (page 920)
`+ setWithObject:` (page 920)
`+ setWithObjects:` (page 921)
`- setByAddingObject:` (page 933)
`- setByAddingObjectsFromSet:` (page 934)
`- setByAddingObjectsFromArray:` (page 933)

Declared In

`NSSet.h`

setWithSet:

Creates and returns a set containing the objects from another set.

```
+ (id)setWithSet:(NSSet *)aSet
```

Parameters

aSet

A set containing the objects to add to the new set. Each object receives a `retain` (page 1312) message as it is added to the new set.

Return Value

A new set containing the objects from *aSet*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ set` (page 919)
`+ setWithArray:` (page 920)
`+ setWithObject:` (page 920)
`+ setWithObjects:` (page 921)
`- setByAddingObject:` (page 933)
`- setByAddingObjectsFromSet:` (page 934)
`- setByAddingObjectsFromArray:` (page 933)

Declared In
NSSet.h

Instance Methods

addObserver:forKeyPath:options:context:

Raises an exception.

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options context:(void *)context
```

Parameters

observer
The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 1287).

keyPath
The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options
A combination of the `NSKeyValueObservingOptions` (page 1291) values that specifies what is included in observation notifications. For possible values, see `NSKeyValueObservingOptions`.

context
Arbitrary data that is passed to *observer* in `observeValueForKeyPath:ofObject:change:context:` (page 1287).

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- removeObserver:forKeyPath:` (page 932)

Declared In
NSKeyValueObserving.h

allObjects

Returns an array containing the receiver's members, or an empty array if the receiver has no members.

```
- (NSArray *)allObjects
```

Return Value
An array containing the receiver's members, or an empty array if the receiver has no members. The order of the objects in the array isn't defined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `anyObject` (page 924)
- `objectEnumerator` (page 931)

Declared In
`NSSet.h`

anyObject

Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.

```
- (id)anyObject
```

Return Value
One of the objects in the receiver, or `nil` if the receiver contains no objects. The object returned is chosen at the receiver's convenience—the selection is not guaranteed to be random.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `allObjects` (page 923)
- `objectEnumerator` (page 931)

Declared In
`NSSet.h`

containsObject:

Returns a Boolean value that indicates whether a given object is present in the receiver.

```
- (BOOL)containsObject:(id)anObject
```

Parameters
anObject
The object for which to test membership of the receiver.

Return Value
`YES` if *anObject* is present in the receiver, otherwise `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `member:` (page 931)

Declared In
`NSSet.h`

count

Returns the number of members in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of members in the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSSet.h

description

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- descriptionWithLocale: (page 925)

Declared In
NSSet.h

descriptionWithLocale:

Returns a string that represents the contents of the receiver, formatted as a property list.

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters
locale

In Mac OS X v10.4 and earlier, this must be a dictionary that specifies options used for formatting each of the receiver's members. In Mac OS X v10.5 and later, you can use an NSLocale object. If you do not want the receiver's members to be formatted, specify nil.

Return Value
A string that represents the contents of the receiver, formatted as a property list.

Discussion
This method sends each of the receiver's members descriptionWithLocale: with *locale* passed as the sole parameter. If the receiver's members do not respond to descriptionWithLocale:, this method sends description (page 1305) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- description (page 925)

Declared In
NSSet.h

initWithArray:

Initializes a newly allocated set with the objects that are contained in a given array.

```
- (id)initWithArray:(NSArray *)array
```

Parameters
array

An array of objects to add to the new set. If the same object appears more than once in *array*, it is represented only once in the returned set. Each object receives a retain (page 1312) message as it is added to the set.

Return Value
An initialized object, which might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithObjects: (page 926)
- initWithObjects:count: (page 927)
- initWithSet: (page 927)
- initWithSet:copyItems: (page 928)
+ setWithArray: (page 920)

Declared In
NSSet.h

initWithObjects:

Initializes a newly allocated set with members taken from the specified list of objects.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters
anObject

The first object to add to the new set.

...

A comma-separated list of objects, ending with nil, to add to the new set. If the same object appears more than once in the list, it is represented only once in the returned set. Each object receives a retain (page 1312) message as it is added to the set

Return Value
An initialized object, which might be different than the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithArray: (page 926)
- initWithObjects:count: (page 927)
- initWithSet: (page 927)
- initWithSet:copyItems: (page 928)
+ setWithObjects: (page 921)

Declared In
NSSet.h

initWithObjects:count:

Initializes a newly allocated set with a specified number of objects from a given C array of objects.

```
- (id)initWithObjects:(id *)objects count:(NSUInteger)count
```

Parameters

objects

A C array of objects to add to the new set. If the same object appears more than once *objects*, it is added only once to the returned set. Each object receives a retain (page 1312) message as it is added to the set.

count

The number of objects from *objects* to add to the new set.

Return Value
An initialized object, which might be different than the original receiver.

Discussion
This method is the designated initializer for NSSet.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithArray: (page 926)
- initWithObjects: (page 926)
- initWithSet: (page 927)
- initWithSet:copyItems: (page 928)
+ setWithObjects:count: (page 921)

Declared In
NSSet.h

initWithSet:

Initializes a newly allocated set and adds to it objects from another given set.

```
- (id)initWithSet:(NSSet *)otherSet
```

Parameters

otherSet

A set containing objects to add to the receiver. Each object is retained as it is added to the receiver.

Return Value

An initialized object, which might be different than the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithArray:` (page 926)
- `initWithObjects:` (page 926)
- `initWithObjects:count:` (page 927)
- `initWithSet:copyItems:` (page 928)
+ `setWithSet:` (page 922)

Declared In

`NSSet.h`

initWithSet:copyItems:

Initializes a newly allocated set and adds to it members of another given set.

```
- (id)initWithSet:(NSSet *)otherSet copyItems:(BOOL)flag
```

Parameters

otherSet

A set containing objects to add to the new set.

flag

If `YES`, the members of *otherSet* are copied, and the copies are added to the receiver. If `NO`, the members of *otherSet* are added to the receiver and retained.

Return Value

An initialized object that contains the members of *otherSet*.

This method returns an initialized object, which might be different than the original receiver.

Discussion

Note that, if *flag* is `YES`, `copyWithZone:` (page 1250) is invoked to make copies—thus, the receiver's new member objects may be immutable, even though their counterparts in *otherSet* were mutable. Also, members must conform to the `NSCopying` protocol)

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithArray:` (page 926)
- `initWithObjects:` (page 926)
- `initWithObjects:count:` (page 927)
- `initWithSet:` (page 927)
+ `setWithSet:` (page 922)

Declared In
NSSet.h

intersectsSet:

Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.

```
- (BOOL)intersectsSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to compare the receiver.

Return Value
YES if at least one object in the receiver is also present in *otherSet*, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isEqualToSet: (page 929)
- isSubsetOfSet: (page 930)

Declared In
NSSet.h

isEqualToSet:

Compares the receiver to another set.

```
- (BOOL)isEqualToSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to compare the receiver.

Return Value
YES if the contents of *otherSet* are equal to the contents of the receiver, otherwise NO.

Discussion
Two sets have equal contents if they each have the same number of members and if each member of one set is present in the other.

Availability
Available in iPhone OS 2.0 and later.

See Also
- intersectsSet: (page 929)
- isEqual: (page 1306) (NSObject protocol)
- isSubsetOfSet: (page 930)

Declared In
NSSet.h

isSubsetOfSet:

Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.

```
- (BOOL)isSubsetOfSet:(NSSet *)otherSet
```

Parameters

otherSet

The set with which to compare the receiver.

Return Value

`YES` if every object in the receiver is also present in *otherSet*, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `intersectsSet:` (page 929)
- `isEqualToSet:` (page 929)

Declared In

`NSSet.h`

makeObjectsPerformSelector:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that specifies the message to send to the members of the receiver. The method must not take any arguments. It should not have the side effect of modifying the receiver. This value must not be `NULL`.

Discussion

The message specified by *aSelector* is sent once to each member of the receiver. This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `makeObjectsPerformSelector:withObject:` (page 930)

Declared In

`NSSet.h`

makeObjectsPerformSelector:withObject:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector

A selector that specifies the message to send to the receiver's members. The method must take a single argument of type `id`. The method should not, as a side effect, modify the receiver. The value must not be `NULL`.

anObject

The object to pass as an argument to the method specified by *aSelector*.

Discussion

The message specified by *aSelector* is sent, with *anObject* as the argument, once to each member of the receiver. This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- makeObjectsPerformSelector:` (page 930)

Declared In

`NSSet.h`

member:

Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.

```
- (id)member:(id)anObject
```

Parameters

anObject

The object for which to test for membership of the receiver.

Return Value

If the receiver contains an object equal to *anObject* (as determined by `isEqual:` (page 1306)) then that object (typically this will be *anObject*), otherwise `nil`.

Discussion

If you override `isEqual:`, you must also override the `hash` method for the `member:` method to work on a set of objects of your class.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSSet.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value

An enumerator object that lets you access each object in the receiver.

Discussion

The following code fragment illustrates how you can use this method.

```
NSEnumerator *enumerator = [mySet objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the set's values */
}
```

When this method is used with mutable subclasses of `NSSet`, your code shouldn't modify the receiver during enumeration. If you intend to modify the receiver, use the `allObjects` (page 923) method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- nextObject` (page 341) (`NSEnumerator`)

Declared In

`NSSet.h`

removeObserver:forKeyPath:

Raises an exception.

```
- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
```

Parameters

observer

The object to remove as an observer.

keyPath

A key-path, relative to the receiver, for which *observer* is registered to receive KVO change notifications. This value must not be `nil`.

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- addObserver:forKeyPath:options:context:` (page 923)

Declared In

`NSKeyValueObserving.h`

setByAddingObject:

Returns a new set formed by adding a given object to the collection defined by the receiver.

```
- (NSSet *)setByAddingObject:(id)anObject
```

Parameters

anObject

The object to add to the collection defined by the receiver.

Return Value

A new set formed by adding *anObject* to the collection defined by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ set (page 919)
+ setWithArray: (page 920)
+ setWithObject: (page 920)
+ setWithObjects: (page 921)
- setByAddingObjectsFromSet: (page 934)
- setByAddingObjectsFromArray: (page 933)

Declared In

NSSet.h

setByAddingObjectsFromArray:

Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromArray:(NSArray *)other
```

Parameters

other

The array of objects to add to the collection defined by the receiver.

Return Value

A new set formed by adding the objects in *other* to the collection defined by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ set (page 919)
+ setWithArray: (page 920)
+ setWithObject: (page 920)
+ setWithObjects: (page 921)
- setByAddingObject: (page 933)
- setByAddingObjectsFromSet: (page 934)

Declared In

NSSet.h

setByAddingObjectsFromSet:

Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromSet:(NSSet *)other
```

Parameters

other

The set of objects to add to the collection defined by the receiver.

Return Value

A new set formed by adding the objects in *other* to the collection defined by the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ set (page 919)
+ setWithArray: (page 920)
+ setWithObject: (page 920)
+ setWithObjects: (page 921)
- setByAddingObject: (page 933)
- setByAddingObjectsFromSet: (page 934)

Declared In

NSSet.h

setValue:forKey:

Invokes setValue:forKey: on each of the receiver's members.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value for the property identified by *key*.

key

The name of one of the properties of the receiver's members.

Availability

Available in iPhone OS 2.0 and later.

See Also

- valueForKey: (page 934)

Declared In

NSKeyValueCoding.h

valueForKey:

Return a set containing the results of invoking valueForKey: on each of the receiver's members.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The name of one of the properties of the receiver's members.

Return Value

A set containing the results of invoking `valueForKey:` (with the argument *key*) on each of the receiver's members.

Discussion

The returned set might not have the same number of members as the receiver. The returned set will not contain any elements corresponding to instances of `valueForKey:` returning `nil` (note that this is in contrast with `NSArray`'s implementation, which may put `NSNull` values in the arrays it returns).

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setValue:forKey:` (page 934)

Declared In

`NSKeyValueCoding.h`

NSSortDescriptor Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSSortDescriptor.h
Companion guide:	Sort Descriptor Programming Topics

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An instance of `NSSortDescriptor` describes a basis for ordering objects by specifying the property to use to compare the objects, the method to use to compare the properties, and whether the comparison should be ascending or descending. Instances of `NSSortDescriptor` are immutable.

You construct an instance of `NSSortDescriptor` by specifying the key path of the property to be compared, the order of the sort (ascending or descending), and (optionally) a selector to use to perform the comparison. The three-argument constructor allows you to specify other comparison selectors such as `caseInsensitiveCompare:` and `localizedCompare:`. Sorting raises an exception if the objects to be sorted do not respond to the sort descriptor's comparison selector.

Note: Many of the descriptions of `NSSortDescriptor` methods refer to "property key". This, briefly, is a string (key) that identifies a property (an attribute or relationship) of an object. You can find a discussion of this terminology in "Object Modeling" in *Cocoa Fundamentals Guide* and in *Key-Value Coding Programming Guide*.

There are a number of situations in which you can use sort descriptors, for example:

- To sort an array (an instance of `NSArray` or `NSMutableArray`—see `sortedArrayUsingDescriptors:` and `sortUsingDescriptors:`)
- To directly compare two objects (see compareObject:toObject: (page 939))
- To specify how the elements in a table view should be arranged (see `sortDescriptors`)
- To specify how the elements managed by an array controller should be arranged (see `sortDescriptors`)
- If you are using Core Data, to specify the ordering of objects returned from a fetch request (see `sortDescriptors`)

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

Tasks

Initializing a Sort Descriptor

`- initWithKey:ascending:` (page 940)
Returns an `NSSortDescriptor` object initialized with a given property key path and sort order, and with the default comparison selector.

`- initWithKey:ascending:selector:` (page 940)
Returns an `NSSortDescriptor` object initialized with a given property key path, sort order, and comparison selector.

Getting Information About a Sort Descriptor

`- ascending` (page 939)
Returns a Boolean value that indicates whether the receiver specifies sorting in ascending order.

- key (page 941)
  Returns the receiver's property key path.
- selector (page 942)
  Returns the selector the receiver specifies to use when comparing objects.

Using Sort Descriptors

- compareObject:toObject: (page 939)
  Returns an NSComparisonResult value that indicates the ordering of two given objects.
- reversedSortDescriptor (page 941)
  Returns a copy of the receiver with the sort order reversed.

Instance Methods

ascending

Returns a Boolean value that indicates whether the receiver specifies sorting in ascending order.

```
- (BOOL)ascending
```

Return Value

YES if the receiver specifies sorting in ascending order, otherwise NO.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSSortDescriptor.h

compareObject:toObject:

Returns an NSComparisonResult value that indicates the ordering of two given objects.

```
- (NSComparisonResult)compareObject:(id)object1 toObject:(id)object2
```

Parameters

object1
  The object to compare with *object2*. This object must have a property accessible using the key-path specified by key (page 941).

  This value must not be nil. If the value is nil, the behavior is undefined and may change in future versions of Mac OS X.

object2
  The object to compare with *object1*. This object must have a property accessible using the key-path specified by key (page 941).

  This value must not be nil. If the value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
`NSOrderedAscending` if *`object1`* is less than *`object2`*, `NSOrderedDescending` if *`object1`* is greater than *`object2`*, or `NSOrderedSame` if *`object1`* is equal to *`object2`*.

Discussion
The ordering is determined by comparing, using the selector specified selector (page 942), the values of the properties specified by key (page 941) of *`object1`* and *`object2`*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSSortDescriptor.h`

initWithKey:ascending:

Returns an `NSSortDescriptor` object initialized with a given property key path and sort order, and with the default comparison selector.

```
- (id)initWithKey:(NSString *)keyPath ascending:(BOOL)ascending
```

Parameters
`keyPath`

The property key to use when performing a comparison. In the comparison, the property is accessed using key-value coding (see *Key-Value Coding Programming Guide*).

`ascending`

`YES` if the receiver specifies sorting in ascending order, otherwise `NO`.

Return Value
An `NSSortDescriptor` object initialized with the property key path specified by *`keyPath`*, sort order specified by *`ascending`*, and the default comparison selector (`compare:`).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithKey:ascending:selector:` (page 940)

Declared In
`NSSortDescriptor.h`

initWithKey:ascending:selector:

Returns an `NSSortDescriptor` object initialized with a given property key path, sort order, and comparison selector.

```
- (id)initWithKey:(NSString *)keyPath ascending:(BOOL)ascending
    selector:(SEL)selector
```

Parameters
`keyPath`

The property key to use when performing a comparison. In the comparison, the property is accessed using key-value coding (see *Key-Value Coding Programming Guide*).

ascending
YES if the receiver specifies sorting in ascending order, otherwise NO.

selector
The method to use when comparing the properties of objects, for example caseInsensitiveCompare: or localizedCompare:. The selector must specify a method implemented by the value of the property identified by *keyPath*. The selector used for the comparison is passed a single parameter, the object to compare against self, and must return the appropriate NSComparisonResult constant. The selector must have the same method signature as:

```
- (NSComparisonResult)localizedCompare:(NSString *)aString
```

Return Value
An NSSortDescriptor object initialized with the property key path specified by *keyPath*, sort order specified by *ascending*, and the selector specified by *selector*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithKey:ascending: (page 940)

Declared In
NSSortDescriptor.h

key

Returns the receiver's property key path.

```
- (NSString *)key
```

Return Value
The receiver's property key path.

Discussion
This key path specifies the property that is compared during sorting.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSSortDescriptor.h

reversedSortDescriptor

Returns a copy of the receiver with the sort order reversed.

```
- (id)reversedSortDescriptor
```

Return Value
A copy of the receiver with the sort order reversed

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSSortDescriptor.h

selector

Returns the selector the receiver specifies to use when comparing objects.

```
- (SEL)selector
```

Return Value
The selector the receiver specifies to use when comparing objects.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSSortDescriptor.h

NSStream Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSStream.h
Companion guide:	Stream Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSStream` is an abstract class for objects representing streams. Its interface is common to all Cocoa stream classes, including its concrete subclasses `NSInputStream` and `NSOutputStream`.

`NSStream` objects provide an easy way to read and write data to and from a variety of media in a device-independent way. You can create stream objects for data located in memory, in a file, or on a network (using sockets), and you can use stream objects without loading all of the data into memory at once.

By default, `NSStream` instances that are not file-based are non-seekable, one-way streams (although custom seekable subclasses are possible). Once the data has been provided or consumed, the data cannot be retrieved from the stream.

Subclassing Notes

`NSStream` is an abstract class, incapable of instantiation and intended to be subclassed. It publishes a programmatic interface that all subclasses must adopt and provide implementations for. The two Apple-provided concrete subclasses of `NSStream`, `NSInputStream` and `NSOutputStream`, are suitable for most purposes. However, there might be situations when you want a peer subclass to `NSInputStream` and `NSOutputStream`. For example, you might want a class that implements a full-duplex (two-way) stream, or a class whose instances are capable of seeking through a stream.

Methods to Override

All subclasses must fully implement the following methods, which are presented in functional pairs:

- `open` (page 946) and `close` (page 946)

  Implement `open` to open the stream for reading or writing and make the stream available to the client directly or, if the stream object is scheduled on a run loop, to the delegate. Implement `close` to close the stream and remove the stream object from the run loop, if necessary. A closed stream should still be able to accept new properties and report its current properties. Once a stream is closed, it cannot be reopened.

- `delegate` (page 946) and `setDelegate:` (page 948)

  Return and set the delegate. By a default, a stream object must be its own delegate; so a `setDelegate:` message with an argument of `nil` should restore this delegate. Do not retain the delegate to prevent retain cycles.

  To learn about delegates and delegation, read "Delegates and Data Sources" in *Cocoa Fundamentals Guide*.

- `scheduleInRunLoop:forMode:` (page 948) and `removeFromRunLoop:forMode:` (page 947)

  Implement `scheduleInRunLoop:forMode:` to schedule the stream object on the specified run loop for the specified mode. Implement `removeFromRunLoop:forMode:` to remove the object from the run loop. See the documentation of the `NSRunLoop` class for details. Once the stream object for an open stream is scheduled on a run loop, it is the responsibility of the subclass as it processes stream data to send `stream:handleEvent:` (page 950) messages to its delegate.

- `propertyForKey:` (page 947) and `setProperty:forKey:` (page 949)

  Implement these methods to return and set, respectively, the property value for the specified key. You may add custom properties, but be sure to handle all properties defined by `NSStream` as well.

- `streamStatus` (page 949) and `streamError` (page 949)

  Implement `streamStatus` to return the current status of the stream as a `NSStreamStatus` constant; you may define new `NSStreamStatus` constants, but be sure to handle the `NSStream`-defined constants properly. Implement`streamError` to return an `NSError` object representing the current error. You might decide to return a custom `NSError` object that can provide complete and localized information about the error.

Tasks

Configuring Streams

- `propertyForKey:` (page 947)
  Returns the receiver's property for a given key.
- `setProperty:forKey:` (page 949)
  Attempts to set the value of a given property of the receiver and returns a Boolean value that indicates whether the value is accepted by the receiver.
- `delegate` (page 946)
  Returns the receiver's delegate.
- `setDelegate:` (page 948)
  Sets the receiver's delegate.

Using Streams

- `open` (page 946)
  Opens the receiving stream.
- `close` (page 946)
  Closes the receiver.
- `stream:handleEvent:` (page 950) *delegate method*
  The delegate receives this message when a given event has occurred on a given stream.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 948)
  Schedules the receiver on a given run loop in a given mode.
- `removeFromRunLoop:forMode:` (page 947)
  Removes the receiver from a given run loop running in a given mode.

Getting Stream Information

- `streamStatus` (page 949)
  Returns the receiver's status.
- `streamError` (page 949)
  Returns an `NSError` object representing the stream error.

Instance Methods

close

Closes the receiver.

```
- (void)close
```

Discussion

Closing the stream terminates the flow of bytes and releases system resources that were reserved for the stream when it was opened. If the stream has been scheduled on a run loop, closing the stream implicitly removes the stream from the run loop. A stream that is closed can still be queried for its properties.

Availability

Available in iPhone OS 2.0 and later.

See Also

- open (page 946)

Declared In

NSStream.h

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Return Value

The receiver's delegate.

Discussion

By default, a stream is its own delegate, and subclasses of NSInputStream and NSOutputStream must maintain this contract.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setDelegate: (page 948)

Declared In

NSStream.h

open

Opens the receiving stream.

```
- (void)open
```

Discussion
A stream must be created before it can be opened. Once opened, a stream cannot be closed and reopened.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `close` (page 946)

Declared In
`NSStream.h`

propertyForKey:
Returns the receiver's property for a given key.

```
- (id)propertyForKey:(NSString *)key
```

Parameters
key

The key for one of the receiver's properties. See "Constants" (page 950) for a description of the available property-key constants and associated values.

Return Value
The receiver's property for the key *key*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setProperty:forKey:` (page 949)

Declared In
`NSStream.h`

removeFromRunLoop:forMode:
Removes the receiver from a given run loop running in a given mode.

```
- (void)removeFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters
aRunLoop

The run loop on which the receiver was scheduled.

mode

The mode for the run loop.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `scheduleInRunLoop:forMode:` (page 948)

Declared In
NSStream.h

scheduleInRunLoop:forMode:

Schedules the receiver on a given run loop in a given mode.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop
The run loop on which to schedule the receiver.

mode
The mode for the run loop.

Discussion
Unless the client is polling the stream, it is responsible for ensuring that the stream is scheduled on at least one run loop and that at least one of the run loops on which the stream is scheduled is being run.

Availability
Available in iPhone OS 2.0 and later.

See Also
- removeFromRunLoop:forMode: (page 947)

Declared In
NSStream.h

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate
The delegate for the receiver.

Discussion
By default, a stream is its own delegate, and subclasses of NSInputStream and NSOutputStream must maintain this contract. If you override this method in a subclass, passing nil must restore the receiver as its own delegate. Delegates are not retained.

To learn about delegates and delegation, read "Delegates and Data Sources" in *Cocoa Fundamentals Guide*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- delegate (page 946)

Declared In
NSStream.h

setProperty:forKey:

Attempts to set the value of a given property of the receiver and returns a Boolean value that indicates whether the value is accepted by the receiver.

```
- (BOOL)setProperty:(id)property forKey:(NSString *)key
```

Parameters

property
The value for *key*.

key
The key for one of the receiver's properties. See "Constants" (page 950) for a description of the available property-key constants and expected values.

Return Value
YES if the value is accepted by the receiver, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- propertyForKey: (page 947)

Declared In
NSStream.h

streamError

Returns an NSError object representing the stream error.

```
- (NSError *)streamError
```

Return Value
An NSError object representing the stream error, or nil if no error has been encountered.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSStream.h

streamStatus

Returns the receiver's status.

```
- (NSStreamStatus)streamStatus
```

Return Value
The receiver's status.

Discussion
See "Constants" (page 950) for a description of the available `NSStreamStatus` constants.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSStream.h`

Delegate Methods

stream:handleEvent:

The delegate receives this message when a given event has occurred on a given stream.

```
- (void)stream:(NSStream *)theStream handleEvent:(NSStreamEvent)streamEvent
```

Parameters

theStream
The stream on which *streamEvent* occurred.

streamEvent
The stream event that occurred,

Discussion
The delegate receives this message only if *theStream* is scheduled on a run loop. The message is sent on the stream object's thread. The delegate should examine *streamEvent* to determine the appropriate action it should take.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSStream.h`

Constants

NSStreamStatus

The type declared for the constants listed in "Stream Status Constants" (page 951).

```
typedef NSUInteger NSStreamStatus;
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSStream.h`

Stream Status Constants

These constants are returned by `streamStatus` (page 949).

```
typedef enum {
    NSStreamStatusNotOpen = 0,
    NSStreamStatusOpening = 1,
    NSStreamStatusOpen = 2,
    NSStreamStatusReading = 3,
    NSStreamStatusWriting = 4,
    NSStreamStatusAtEnd = 5,
    NSStreamStatusClosed = 6,
    NSStreamStatusError = 7
};
```

Constants

`NSStreamStatusNotOpen`
The stream is not open for reading or writing.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusOpening`
The stream is being opened for reading or for writing.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusOpen`
The stream is open.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusReading`
Data is being read from the stream.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusWriting`
Data is being written to the stream.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusAtEnd`
There is no more data to read, or no more data can be written to the stream.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusClosed`
The stream is closed.
Available in iPhone OS 2.0 and later.
Declared in `NSStream.h`

`NSStreamStatusError`
An error has occurred on the stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Declared In
`NSStream.h`

NSStreamEvent

The type declared for the constants listed in "Stream Event Constants" (page 952).

```
typedef NSUInteger NSStreamEvent;
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSStream.h`

Stream Event Constants

These constants are sent to the delegate in the second parameter of `stream:handleEvent:` (page 950).

```
typedef enum {
    NSStreamEventNone = 0,
    NSStreamEventOpenCompleted = 1 << 0,
    NSStreamEventHasBytesAvailable = 1 << 1,
    NSStreamEventHasSpaceAvailable = 1 << 2,
    NSStreamEventErrorOccurred = 1 << 3,
    NSStreamEventEndEncountered = 1 << 4
};
```

Constants
`NSStreamEventNone`
No event has occurred.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamEventOpenCompleted`
The open has completed successfully.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamEventHasBytesAvailable`
The stream has bytes to be read.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamEventHasSpaceAvailable`
The stream can accept bytes for writing.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamEventErrorOccurred`
An error has occurred on the stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamEventEndEncountered`
The end of the stream has been reached.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Declared In
`NSStream.h`

NSStream Property Keys

`NSStream` defines these string constants as keys for accessing stream properties using `propertyForKey:` (page 947) and setting properties with `setProperty:forKey:` (page 949):

```
extern NSString * const NSStreamSocketSecurityLevelKey  ;
extern NSString * const NSStreamSocketSecurityLevelNone ;
extern NSString * const NSStreamSocketSecurityLevelSSLv2 ;
extern NSString * const NSStreamSocketSecurityLevelSSLv3 ;
extern NSString * const NSStreamSocketSecurityLevelTLSv1 ;
extern NSString * const NSStreamSocketSecurityLevelNegotiatedSSL;
extern NSString * const NSStreamSOCKSProxyConfigurationKey  ;
extern NSString * const NSStreamSOCKSProxyHostKey  ;
extern NSString * const NSStreamSOCKSProxyPortKey  ;
extern NSString * const NSStreamSOCKSProxyVersionKey  ;
extern NSString * const NSStreamSOCKSProxyUserKey  ;
extern NSString * const NSStreamSOCKSProxyPasswordKey ;
extern NSString * const NSStreamSOCKSProxyVersion4  ;
extern NSString * const NSStreamSOCKSProxyVersion5  ;
extern NSString * const NSStreamDataWrittenToMemoryStreamKey ;
extern NSString * const NSStreamFileCurrentOffsetKey  ;
```

Constants
`NSStreamSocketSecurityLevelKey`
The security level of the target stream. May be one of the following values: `NSStreamSocketSecurityLevelNone`, `NSStreamSocketSecurityLevelSSLv2`, `NSStreamSocketSecurityLevelSSLv3`, `NSStreamSocketSecurityLevelTLSv1`, or `NSStreamSocketSecurityLevelNegotiatedSSL`.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyConfigurationKey`
Value is an `NSDictionary` object containing SOCKS proxy configuration information.

The dictionary returned from the System Configuration framework for SOCKS proxies usually suffices.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamDataWrittenToMemoryStreamKey`
Value is an `NSData` instance containing the data written to a memory stream.

Use this property when you have an output-stream object instantiated to collect written data in memory. The value of this property is read-only.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamFileCurrentOffsetKey`
Value is an `NSNumber` object containing the current absolute offset of the stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Declared In
`NSStream.h`

NSStream Error Domains

`NSStream` defines these string constants to represent error domains that can be returned by `streamError` (page 949):

```
extern NSString * const NSStreamSocketSSLErrorDomain  ;
extern NSString * const NSStreamSOCKSErrorDomain  ;
```

Constants
`NSStreamSocketSSLErrorDomain`
The error domain used by `NSError` when reporting SSL errors.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSErrorDomain`
The error domain used by `NSError` when reporting SOCKS errors.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Declared In
`NSStream.h`

Secure-Socket Layer (SSL) Security Level

`NSStream` defines these string constants for specifying the secure-socket layer (SSL) security level.

```
NSString * const NSStreamSocketSecurityLevelNone;
NSString * const NSStreamSocketSecurityLevelSSLv2;
NSString * const NSStreamSocketSecurityLevelSSLv3;
NSString * const NSStreamSocketSecurityLevelTLSv1;
NSString * const NSStreamSocketSecurityLevelNegotiatedSSL
```

Constants

`NSStreamSocketSecurityLevelNone`

Specifies that no security level be set for a socket stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSocketSecurityLevelSSLv2`

Specifies that SSL version 2 be set as the security protocol for a socket stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSocketSecurityLevelSSLv3`

Specifies that SSL version 3 be set as the security protocol for a socket stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSocketSecurityLevelTLSv1`

Specifies that TLS version 1 be set as the security protocol for a socket stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSocketSecurityLevelNegotiatedSSL`

Specifies that the highest level security protocol that can be negotiated be set as the security protocol for a socket stream.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Discussion

You access and set these values using the `NSStreamSocketSecurityLevelKey` property key.

Declared In

`NSStream.h`

SOCKS Proxy Configuration Values

`NSStream` defines these string constants for use as keys to specify SOCKS proxy configuration values in an `NSDictionary` object.

```
NSString * const NSStreamSOCKSProxyHostKey;
NSString * const NSStreamSOCKSProxyPortKey;
NSString * const NSStreamSOCKSProxyVersionKey;
NSString * const NSStreamSOCKSProxyUserKey;
NSString * const NSStreamSOCKSProxyPasswordKey;
NSString * const NSStreamSOCKSProxyVersion4;
NSString * const NSStreamSOCKSProxyVersion5
```

Constants

`NSStreamSOCKSProxyHostKey`

Value is an `NSString` object that represents the SOCKS proxy host.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyPortKey`

Value is an `NSNumber` object containing an integer that represents the port on which the proxy listens.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyVersionKey`

Value is either `NSStreamSOCKSProxyVersion4` or `NSStreamSOCKSProxyVersion5`.

If this key is not present, `NSStreamSOCKSProxyVersion5` is used by default.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyUserKey`

Value is an `NSString` object containing the user's name.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyPasswordKey`

Value is an `NSString` object containing the user's password.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyVersion4`

Possible value for `NSStreamSOCKSProxyVersionKey`.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

`NSStreamSOCKSProxyVersion5`

Possible value for `NSStreamSOCKSProxyVersionKey`.

Available in iPhone OS 2.0 and later.

Declared in `NSStream.h`

Discussion

You set the dictionary object as the current SOCKS proxy configuration using the `NSStreamSOCKSProxyConfigurationKey` key

Declared In

`NSStream.h`

NSString Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSMutableCopying NSObject (`NSObject`)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSString.h Foundation/NSPathUtilities.h Foundation/NSURL.h
Companion guides:	String Programming Guide for Cocoa Property List Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSString` class declares the programmatic interface for an object that manages immutable strings. (An immutable string is a text string that is defined when it is created and subsequently cannot be changed. `NSString` is implemented to represent an array of Unicode characters (in other words, a text string).

The mutable subclass of NSString is NSMutableString.

The NSString class has two primitive methods—length (page 1011) and characterAtIndex: (page 980)—that provide the basis for all other methods in its interface. The length (page 1011) method returns the total number of Unicode characters in the string. characterAtIndex: (page 980) gives access to each character in the string by index, with index values starting at 0.

NSString declares methods for finding and comparing strings. It also declares methods for reading numeric values from strings, for combining strings in various ways, and for converting a string to different forms (such as encoding and case changes).

The Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the NSAttributedString classes. Additionally, methods to support string drawing are described in NSString Additions, found in the Application Kit.

NSString is "toll-free bridged" with its Core Foundation counterpart, CFString (see CFStringRef). This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an NSString * parameter, you can pass a CFStringRef, and in a function where you see a CFStringRef parameter, you can pass an NSString instance (you cast one type to the other to suppress compiler warnings). This also applies to your concrete subclasses of NSString. See Interchangeable Data Types for more information on toll-free bridging.

String Objects

NSString objects represent character strings in frameworks. Representing strings as objects allows you to use strings wherever you use other objects. It also provides the benefits of encapsulation, so that string objects can use whatever encoding and storage are needed for efficiency while simply appearing as arrays of characters. The cluster's two public classes, NSString and NSMutableString, declare the programmatic interface for non-editable and editable strings, respectively.

Note: An immutable string is a text string that is defined when it is created and subsequently cannot be changed. An immutable string is implemented as an array of Unicode characters (in other words, a text string). To create and manage an immutable string, use the NSString class. To construct and manage a string that can be changed after it has been created, use NSMutableString.

The objects you create using NSString and NSMutableString are referred to as string objects (or, when no confusion will result, merely as strings). The term C string refers to the standard char * type. Because of the nature of class clusters, string objects aren't actual instances of the NSString or NSMutableString classes but of one of their private subclasses. Although a string object's class is private, its interface is public, as declared by these abstract superclasses, NSString and NSMutableString. The string classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert a string of one type to the other.

A string object presents itself as an array of Unicode characters (Unicode is a registered trademark of Unicode, Inc.). You can determine how many characters a string object contains with the length (page 1011) method and can retrieve a specific character with the characterAtIndex: (page 980) method. These two "primitive" methods provide basic access to a string object. Most use of strings, however, is at a higher level, with the strings being treated as single entities: You compare strings against one another, search them for substrings, combine them into new strings, and so on. If you

need to access string objects character by character, you must understand the Unicode character encoding, specifically issues related to composed character sequences. For details see *The Unicode Standard, Version 4.0* (The Unicode Consortium, Boston: Addison-Wesley, 2003, ISBN 0-321-18578-1) and the Unicode Consortium web site: http://www.unicode.org/.

When creating an `NSString` object from a UTF-16-encoded string (or a byte stream interpreted as UTF-16), if the byte order is not otherwise specified, `NSString` assumes that the UTF-16 characters are big-endian, unless there is a BOM (byte-order mark), in which case the BOM dictates the byte order. When creating an `NSString` object from an array of Unicode characters, the returned string is always native-endian, since the array always contains Unicode characters in native byte order.

Over distributed-object connections, mutable string objects are passed by-reference and immutable string objects are passed by-copy.

Subclassing Notes

It is possible to subclass `NSString` (and `NSMutableString`), but doing so requires providing storage facilities for the string (which is not inherited by subclasses) and implementing two primitive methods. The abstract `NSString` and `NSMutableString` classes are the public interface of a class cluster consisting mostly of private, concrete classes that create and return a string object appropriate for a given situation. Making your own concrete subclass of this cluster imposes certain requirements (discussed in "Methods to Override" (page 959)).

Make sure your reasons for subclassing `NSString` so are valid. Instances of your subclass should represent a string and not something else. Thus the only attributes the subclass should have are the length of the character buffer it's managing and access to individual characters in the buffer. Valid reasons for making a subclass of `NSString` include providing a different backing store (perhaps for better performance) or implementing some aspect of object behavior differently, such as memory management. If your purpose is to add non-essential attributes or metadata to your subclass of `NSString`, a better alternative would be object composition (see "Alternatives to Subclassing" (page 960)). Cocoa already provides an example of this with the `NSAttributedString` class.

Methods to Override

Any subclass of `NSString` *must* override the primitive instance methods `length` (page 1011) and `characterAtIndex:` (page 980). These methods must operate on the backing store that you provide for the characters of the string. For this backing store you can use a static array, a dynamically allocated buffer, a standard `NSString` object, or some other data type or mechanism. You may also choose to override, partially or fully, any other `NSString` method for which you want to provide an alternative implementation. For example, for better performance it is recommended that you override `getCharacters:range:` (page 992) and give it a faster implementation.

You might want to implement an initializer for your subclass that is suited to the backing store that the subclass is managing. The `NSString` class does not have a designated initializer, so your initializer need only invoke the `init` (page 803) method of `super`. The `NSString` class adopts the `NSCopying`, `NSMutableCopying`, and `NSCoding` protocols; if you want instances of your own custom subclass created from copying or coding, override the methods in these protocols.

Note that you shouldn't override the `hash` (page 996) method.

Alternatives to Subclassing

Often a better and easier alternative to making a subclass of `NSString`—or of any other abstract, public class of a class cluster, for that matter—is object composition. This is especially the case when your intent is to add to the subclass metadata or some other attribute that is not essential to a string object. In object composition, you would have an `NSString` object as one instance variable of your custom class (typically a subclass of `NSObject`) and one or more instance variables that store the metadata that you want for the custom object. Then just design your subclass interface to include accessor methods for the embedded string object and the metadata.

If the behavior you want to add supplements that of the existing class, you could write a category on `NSString`. Keep in mind, however, that this category will be in effect for all instances of `NSString` that you use, and this might have unintended consequences.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

NSMutableCopying
- `mutableCopyWithZone:` (page 1300)

Tasks

Creating and Initializing Strings

`+ string` (page 972)
Returns an empty string.

`- init` (page 998)
Returns an initialized `NSString` object that contains no characters.

`- initWithBytes:length:encoding:` (page 998)
Returns an initialized `NSString` object containing a given number of bytes from a given C array of bytes in a given encoding.

`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 999)
Returns an initialized `NSString` object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

`- initWithCharacters:length:` (page 999)
Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

- initWithCharactersNoCopy:length:freeWhenDone: (page 1000)
  Returns an initialized NSString object that contains a given number of characters from a given C array of Unicode characters.
- initWithString: (page 1007)
  Returns an NSString object initialized by copying the characters from another given string.
- initWithCString:encoding: (page 1003)
  Returns an NSString object initialized using the characters in a given C array, interpreted according to a given encoding.
- initWithUTF8String: (page 1008)
  Returns an NSString object initialized by copying the characters a given C array of UTF8-encoded bytes.
- initWithFormat: (page 1004)
  Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted.
- initWithFormat:arguments: (page 1005)
  Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
- initWithFormat:locale: (page 1006)
  Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- initWithFormat:locale:arguments: (page 1006)
  Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.
- initWithData:encoding: (page 1004)
  Returns an NSString object initialized by converting given data into Unicode characters using a given encoding.
+ stringWithFormat: (page 976)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted.
+ localizedStringWithFormat: (page 970)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.
+ stringWithCharacters:length: (page 972)
  Returns a string containing a given number of characters taken from a given C array of Unicode characters.
+ stringWithString: (page 976)
  Returns a string created by copying the characters from another given string.
+ stringWithCString:encoding: (page 975)
  Returns a string containing the bytes in a given C array, interpreted according to a given encoding.
+ stringWithUTF8String: (page 977)
  Returns a string created by copying the data from a given C array of UTF8-encoded bytes.

Creating and Initializing a String from a File

`+ stringWithContentsOfFile:encoding:error:` (page 973)
Returns a string created by reading data from the file at a given path interpreted using a given encoding.

`- initWithContentsOfFile:encoding:error:` (page 1001)
Returns an `NSString` object initialized by reading data from the file at a given path using a given encoding.

`+ stringWithContentsOfFile:usedEncoding:error:` (page 973)
Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

`- initWithContentsOfFile:usedEncoding:error:` (page 1001)
Returns an `NSString` object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

Creating and Initializing a String from an URL

`+ stringWithContentsOfURL:encoding:error:` (page 974)
Returns a string created by reading data from a given URL interpreted using a given encoding.

`- initWithContentsOfURL:encoding:error:` (page 1002)
Returns an `NSString` object initialized by reading data from a given URL interpreted using a given encoding.

`+ stringWithContentsOfURL:usedEncoding:error:` (page 975)
Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.

`- initWithContentsOfURL:usedEncoding:error:` (page 1002)
Returns an `NSString` object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

Writing to a File or URL

`- writeToFile:atomically:encoding:error:` (page 1040)
Writes the contents of the receiver to a file at a given path using a given encoding.

`- writeToURL:atomically:encoding:error:` (page 1041)
Writes the contents of the receiver to the URL specified by *`url`* using the specified encoding.

Getting a String's Length

`- length` (page 1011)
Returns the number of Unicode characters in the receiver.

`- lengthOfBytesUsingEncoding:` (page 1012)
Returns the number of bytes required to store the receiver in a given encoding.

`- maximumLengthOfBytesUsingEncoding:` (page 1015)
Returns the maximum number of bytes needed to store the receiver in a given encoding.

Getting Characters and Bytes

- `characterAtIndex:` (page 980)
  Returns the character at a given array position.
- `getCharacters:` (page 992)
  Returns by reference the characters from the receiver.
- `getCharacters:range:` (page 992)
  Copies characters from a given range in the receiver into a given buffer.
- `getBytes:maxLength:usedLength:encoding:options:range:remainingRange:` (page 991)
  Gets a given range of characters as bytes in a specified encoding.

Getting C Strings

- `cStringUsingEncoding:` (page 986)
  Returns a representation of the receiver as a C string using a given encoding.
- `getCString:maxLength:encoding:` (page 993)
  Converts the receiver's content to a given encoding and stores them in a buffer.
- `UTF8String` (page 1040)
  Returns a null-terminated UTF8 representation of the receiver.

Combining Strings

- `stringByAppendingFormat:` (page 1027)
  Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
- `stringByAppendingString:` (page 1029)
  Returns a new string made by appending a given string to the receiver.
- `stringByPaddingToLength:withString:startingAtIndex:` (page 1032)
  Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

Dividing Strings

- `componentsSeparatedByString:` (page 985)
  Returns an array containing substrings from the receiver that have been divided by a given separator.
- `componentsSeparatedByCharactersInSet:` (page 985)
  Returns an array containing substrings from the receiver that have been divided by characters in a given set.
- `stringByTrimmingCharactersInSet:` (page 1037)
  Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
- `substringFromIndex:` (page 1038)
  Returns a new string containing the characters of the receiver from the one at a given index to the end.

- substringWithRange: (page 1039)
  Returns a string object containing the characters of the receiver that lie within a given range.
- substringToIndex: (page 1038)
  Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

Finding Characters and Substrings

- rangeOfCharacterFromSet: (page 1019)
  Returns the range in the receiver of the first character found from a given character set.
- rangeOfCharacterFromSet:options: (page 1020)
  Returns the range in the receiver of the first character found, using given options, from a given character set.
- rangeOfCharacterFromSet:options:range: (page 1020)
  Returns the range in the receiver of the first character found from a given character set found in a given range with given options.
- rangeOfString: (page 1023)
  Returns the range of the first occurrence within the receiver of a given string.
- rangeOfString:options: (page 1023)
  Returns the range of the first occurrence within the receiver of a given string, subject to given options
- rangeOfString:options:range: (page 1024)
  Returns the range of the first occurrence within a given range of the receiver of a given string, subject to given options.
- rangeOfString:options:range:locale: (page 1025)
  Returns the range of the first occurrence within a given range of the receiver of a given string, subject to given options.

Replacing Substrings

- stringByReplacingOccurrencesOfString:withString: (page 1033)
  Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.
- stringByReplacingOccurrencesOfString:withString:options:range: (page 1034)
  Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.
- stringByReplacingCharactersInRange:withString: (page 1033)
  Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

Determining Line and Paragraph Ranges

- getLineStart:end:contentsEnd:forRange: (page 994)
  Returns by reference the beginning of the first line and the end of the last line touched by the given range.

- `lineRangeForRange:` (page 1012)
  Returns the range of characters representing the line or lines containing a given range.
- `getParagraphStart:end:contentsEnd:forRange:` (page 995)
  Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.
- `paragraphRangeForRange:` (page 1015)
  Returns the range of characters representing the paragraph or paragraphs containing a given range.

Determining Composed Character Sequences

- `rangeOfComposedCharacterSequenceAtIndex:` (page 1021)
  Returns the range in the receiver of the composed character sequence located at a given index.
- `rangeOfComposedCharacterSequencesForRange:` (page 1022)
  Returns the range in the receiver of the composed character sequence in a given range.

Converting String Contents Into a Property List

- `propertyList` (page 1018)
  Parses the receiver as a text representation of a property list, returning an `NSString`, `NSData`, `NSArray`, or `NSDictionary` object, according to the topmost element.
- `propertyListFromStringsFileFormat` (page 1019)
  Returns a dictionary object initialized with the keys and values found in the receiver.

Identifying and Comparing Strings

- `caseInsensitiveCompare:` (page 979)
  Returns the result of invoking `compare:options:` (page 981) with `NSCaseInsensitiveSearch` as the only option.
- `localizedCaseInsensitiveCompare:` (page 1013)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.
- `compare:` (page 981)
  Returns the result of invoking `compare:options:range:` (page 982) with no options and the receiver's full extent as the range.
- `localizedCompare:` (page 1013)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and another given string using a localized comparison.
- `compare:options:` (page 981)
  Returns the result of invoking `compare:options:range:` (page 982) with a given mask as the options and the receiver's full extent as the range.
- `compare:options:range:` (page 982)
  Returns the result of invoking `compare:options:range:locale:` (page 983) with a `nil` locale.

- `compare:options:range:locale:` (page 983)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.
- `hasPrefix:` (page 997)
  Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
- `hasSuffix:` (page 997)
  Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
- `isEqualToString:` (page 1010)
  Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.
- `hash` (page 996)
  Returns an unsigned integer that can be used as a hash table address.

Folding Strings

- `stringByFoldingWithOptions:locale:` (page 1032)
  Returns a string with the given character folding options applied.

Getting a Shared Prefix

- `commonPrefixWithString:options:` (page 980)
  Returns a string containing prefix the receiver and a given string have in common.

Changing Case

- `capitalizedString` (page 978)
  Returns a capitalized representation of the receiver.
- `lowercaseString` (page 1014)
  Returns lowercased representation of the receiver.
- `uppercaseString` (page 1039)
  Returns an uppercased representation of the receiver.

Getting Strings with Mapping

- `decomposedStringWithCanonicalMapping` (page 988)
  Returns a string made by normalizing the receiver's contents using Form D.
- `decomposedStringWithCompatibilityMapping` (page 988)
  Returns a string made by normalizing the receiver's contents using Form KD.
- `precomposedStringWithCanonicalMapping` (page 1017)
  Returns a string made by normalizing the receiver's contents using Form C.
- `precomposedStringWithCompatibilityMapping` (page 1018)
  Returns a string made by normalizing the receiver's contents using Form KC.

Getting Numeric Values

- `doubleValue` (page 989)
  Returns the floating-point value of the receiver's text as a `double`.
- `floatValue` (page 990)
  Returns the floating-point value of the receiver's text as a `float`.
- `intValue` (page 1009)
  Returns the integer value of the receiver's text.
- `integerValue` (page 1008)
  Returns the `NSInteger` value of the receiver's text.
- `longLongValue` (page 1014)
  Returns the `long long` value of the receiver's text.
- `boolValue` (page 977)
  Returns the Boolean value of the receiver's text.

Working with Encodings

+ `availableStringEncodings` (page 969)
  Returns a zero-terminated list of the encodings string objects support in the application's environment.
+ `defaultCStringEncoding` (page 969)
  Returns the C-string encoding assumed for any method accepting a C string as an argument.
+ `localizedNameOfStringEncoding:` (page 970)
  Returns a human-readable string giving the name of a given encoding.
- `canBeConvertedToEncoding:` (page 978)
  Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.
- `dataUsingEncoding:` (page 987)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `dataUsingEncoding:allowLossyConversion:` (page 987)
  Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.
- `description` (page 989)
  Returns the receiver.
- `fastestEncoding` (page 989)
  Returns the fastest encoding to which the receiver may be converted without loss of information.
- `smallestEncoding` (page 1025)
  Returns the smallest encoding to which the receiver can be converted without loss of information.

Working with Paths

+ `pathWithComponents:` (page 971)
  Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.

- `pathComponents` (page 1016)
  Returns an array of `NSString` objects containing, in order, each path component of the receiver.

- `completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:` (page 984)
  Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.

- `fileSystemRepresentation` (page 990)
  Returns a file system-specific representation of the receiver.

- `getFileSystemRepresentation:maxLength:` (page 993)
  Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.

- `isAbsolutePath` (page 1010)
  Returning a Boolean value that indicates whether the receiver represents an absolute path.

- `lastPathComponent` (page 1011)
  Returns the last path component of the receiver.

- `pathExtension` (page 1017)
  Interprets the receiver as a path and returns the receiver's extension, if any.

- `stringByAbbreviatingWithTildeInPath` (page 1026)
  Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.

- `stringByAppendingPathComponent:` (page 1027)
  Returns a new string made by appending to the receiver a given string.

- `stringByAppendingPathExtension:` (page 1028)
  Returns a new string made by appending to the receiver an extension separator followed by a given extension.

- `stringByDeletingLastPathComponent` (page 1030)
  Returns a new string made by deleting the last path component from the receiver, along with any final path separator.

- `stringByDeletingPathExtension` (page 1030)
  Returns a new string made by deleting the extension (if any, and only the last) from the receiver.

- `stringByExpandingTildeInPath` (page 1031)
  Returns a new string made by expanding the initial component of the receiver to its full path value.

- `stringByResolvingSymlinksInPath` (page 1035)
  Returns a new string made from the receiver by resolving all symbolic links and standardizing path.

- `stringByStandardizingPath` (page 1036)
  Returns a new string made by removing extraneous path components from the receiver.

- `stringsByAppendingPaths:` (page 1037)
  Returns an array of strings made by separately appending to the receiver each string in in a given array.

Working with URLs

`- stringByAddingPercentEscapesUsingEncoding:` (page 1026)
Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.

`- stringByReplacingPercentEscapesUsingEncoding:` (page 1035)
Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

Class Methods

availableStringEncodings

Returns a zero-terminated list of the encodings string objects support in the application's environment.

```
+ (const NSStringEncoding *)availableStringEncodings
```

Return Value
A zero-terminated list of the encodings string objects support in the application's environment.

Discussion
Among the more commonly used encodings are:

```
NSASCIIStringEncoding
NSUnicodeStringEncoding
NSISOLatin1StringEncoding
NSISOLatin2StringEncoding
NSSymbolStringEncoding
```

See the "Constants" (page 1041) section for a larger list and descriptions of many supported encodings. In addition to those encodings listed here, you can also use the encodings defined for CFString in Core Foundation; you just need to call the `CFStringConvertEncodingToNSStringEncoding` function to convert them to a usable format.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ localizedNameOfStringEncoding:` (page 970)

Declared In
`NSString.h`

defaultCStringEncoding

Returns the C-string encoding assumed for any method accepting a C string as an argument.

```
+ (NSStringEncoding)defaultCStringEncoding
```

Return Value
The C-string encoding assumed for any method accepting a C string as an argument.

Discussion
This method returns a user-dependent encoding who value is derived from user's default language and potentially other factors. You might sometimes need to use this encoding when interpreting user documents with unknown encodings, in the absence of other hints, but in general this encoding should be used rarely, if at all. Note that some potential values might result in unexpected encoding conversions of even fairly straightforward `NSString` content—for example, punctuation characters with a bidirectional encoding.

Methods that accept a C string as an argument use `...CString...` in the keywords for such arguments: for example, `stringWithCString:`—note, though, that these are deprecated. The default C-string encoding is determined from system information and can't be changed programmatically for an individual process. See "String Encodings" (page 1045) for a full list of supported encodings.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

localizedNameOfStringEncoding:

Returns a human-readable string giving the name of a given encoding.

```
+ (NSString *)localizedNameOfStringEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
A string encoding.

Return Value
A human-readable string giving the name of *encoding* in the current locale's language.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

localizedStringWithFormat:

Returns a string created by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
+ (id)localizedStringWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A string created by using *format* as a template into which the following argument values are substituted according to the formatting information to the user's default locale.

Discussion

This method is equivalent to using `initWithFormat:locale:` (page 1006) and passing `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]` as the locale argument.

As an example of formatting, this method replaces the decimal according to the locale in `%f` and `%d` substitutions, and calls `descriptionWithLocale:` instead of `description` where necessary.

This code excerpt creates a string from another string and a `float`:

```
NSString *myString = [NSString localizedStringWithFormat:@"%@:  %f\n", @"Cost",
 1234.56];
```

The resulting string has the value "`Cost: 1234.560000\n`" if the locale is `en_US`, and "`Cost: 1234,560000\n`" if the locale is `fr_FR`.

See Formatting String Objects for more information.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithFormat:` (page 976)

`- initWithFormat:locale:` (page 1006)

Declared In

`NSString.h`

pathWithComponents:

Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.

```
+ (NSString *)pathWithComponents:(NSArray *)components
```

Parameters

components

An array of `NSString` objects representing a file path. To create an absolute path, use a slash mark ("`/`") as the first component. To include a trailing path divider, use an empty string as the last component.

Return Value
A string built from the strings in *components* by concatenating them (in the order they appear in the array) with a path separator between each pair.

Discussion
This method doesn't clean up the path created; use `stringByStandardizingPath` (page 1036) to resolve empty components, references to the parent directory, and so on.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- pathComponents` (page 1016)

Declared In
`NSPathUtilities.h`

string

Returns an empty string.

```
+ (id)string
```

Return Value
An empty string.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- init` (page 998)

Declared In
`NSString.h`

stringWithCharacters:length:

Returns a string containing a given number of characters taken from a given C array of Unicode characters.

```
+ (id)stringWithCharacters:(const unichar *)chars length:(NSUInteger)length
```

Parameters

chars

A C array of Unicode characters; the value must not be `NULL`.

Important: Raises an exception if *chars* is `NULL`, even if *length* is `0`.

length

The number of characters to use from *chars*.

Return Value
A string containing *length* Unicode characters taken (starting with the first) from *chars*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithCharacters:length:` (page 999)

Declared In
`NSString.h`

stringWithContentsOfFile:encoding:error:

Returns a string created by reading data from the file at a given path interpreted using a given encoding.

```
+ (id)stringWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters
path

A path to a file.

enc

The encoding of the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value
A string created by reading data from the file named by *path* using the encoding, *enc*. If the file can't be opened or there is an encoding error, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithContentsOfFile:encoding:error:` (page 1001)

Declared In
`NSString.h`

stringWithContentsOfFile:usedEncoding:error:

Returns a string created by reading data from the file at a given path and returns by reference the encoding used to interpret the file.

```
+ (id)stringWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding
    *)enc error:(NSError **)error
```

Parameters
path

A path to a file.

enc

Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value

A string created by reading data from the file named by *path*. If the file can't be opened or there is an encoding error, returns `nil`.

Discussion

This method attempts to determine the encoding of the file at *path*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithContentsOfFile:encoding:error:` (page 1001)

Declared In

`NSString.h`

stringWithContentsOfURL:encoding:error:

Returns a string created by reading data from a given URL interpreted using a given encoding.

```
+ (id)stringWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url

The URL to read.

enc

The encoding of the data at *url*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value

A string created by reading data from *URL* using the encoding, *enc*. If the URL can't be opened or there is an encoding error, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithContentsOfURL:usedEncoding:error:` (page 975)
`- initWithContentsOfURL:encoding:error:` (page 1002)

Declared In

`NSString.h`

stringWithContentsOfURL:usedEncoding:error:

Returns a string created by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
+ (id)stringWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, you may pass in `NULL`.

Return Value

A string created by reading data from *url*. If the URL can't be opened or there is an encoding error, returns `nil`.

Discussion

This method attempts to determine the encoding at *url*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithContentsOfURL:encoding:error:` (page 974)

`- initWithContentsOfURL:usedEncoding:error:` (page 1002)

Declared In

`NSString.h`

stringWithCString:encoding:

Returns a string containing the bytes in a given C array, interpreted according to a given encoding.

```
+ (id)stringWithCString:(const char *)cString encoding:(NSStringEncoding)enc
```

Parameters

cString

A C array of bytes. The array must end with a `NULL` character; intermediate `NULL` characters are not allowed.

enc

The encoding of *cString*.

Return Value

A string containing the characters described in *cString*.

Discussion

If *cString* is not a `NULL`-terminated C string, or *encoding* does not match the actual encoding, the results are undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithCString:encoding:` (page 1003)

Declared In
`NSString.h`

stringWithFormat:
Returns a string created by using a given format string as a template into which the remaining argument values are substituted.

```
+ (id)stringWithFormat:(NSString *)format, ...
```

Parameters
format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
A string created by using *format* as a template into which the remaining argument values are substituted according to the canonical locale.

Discussion
This method is similar to `localizedStringWithFormat:` (page 970), but using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithFormat:` (page 1004)
+ `localizedStringWithFormat:` (page 970)

Declared In
`NSString.h`

stringWithString:
Returns a string created by copying the characters from another given string.

```
+ (id)stringWithString:(NSString *)aString
```

Parameters

aString

The string from which to copy characters. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

A string created by copying the characters from *aString*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithString:` (page 1007)

Declared In

`NSString.h`

stringWithUTF8String:

Returns a string created by copying the data from a given C array of UTF8-encoded bytes.

```
+ (id)stringWithUTF8String:(const char *)bytes
```

Parameters

bytes

A `NULL`-terminated C array of bytes in UTF8 encoding.

Important: Raises an exception if *bytes* is `NULL`.

Return Value

A string created by copying the data from *bytes*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithString:` (page 1007)

Declared In

`NSString.h`

Instance Methods

boolValue

Returns the Boolean value of the receiver's text.

```
- (BOOL)boolValue
```

Return Value
The Boolean value of the receiver's text. Returns `YES` on encountering one of "Y", "y", "T", "t", or a digit 1-9—the method ignores any trailing characters. Returns `NO` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
The method assumes a decimal representation and skips whitespace at the beginning of the string. It also skips initial whitespace characters, or optional -/+ sign followed by zeroes.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `integerValue` (page 1008)
- `scanInt:` (page 907) (`NSScanner`)

Declared In
`NSString.h`

canBeConvertedToEncoding:
Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.

```
- (BOOL)canBeConvertedToEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
A string encoding.

Return Value
`YES` if the receiver can be converted to *encoding* without loss of information. Returns `NO` if characters would have to be changed or deleted in the process of changing encodings.

Discussion
If you plan to actually convert a string, the `dataUsingEncoding:...` methods return `nil` on failure, so you can avoid the overhead of invoking this method yourself by simply trying to convert the string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `dataUsingEncoding:allowLossyConversion:` (page 987)

Declared In
`NSString.h`

capitalizedString
Returns a capitalized representation of the receiver.

```
- (NSString *)capitalizedString
```

Return Value
A string with the first character from each word in the receiver changed to its corresponding uppercase value, and all remaining characters set to their corresponding lowercase values.

Discussion
A "word" here is any sequence of characters delimited by spaces, tabs, or line terminators (listed under `getLineStart:end:contentsEnd:forRange:` (page 994)). Other common word delimiters such as hyphens and other punctuation aren't considered, so this method may not generally produce the desired results for multiword strings.

Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See `lowercaseString` (page 1014) for an example.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `lowercaseString` (page 1014)
- `uppercaseString` (page 1039)

Declared In
`NSString.h`

caseInsensitiveCompare:
Returns the result of invoking `compare:options:` (page 981) with `NSCaseInsensitiveSearch` as the only option.

```
- (NSComparisonResult)caseInsensitiveCompare:(NSString *)aString
```

Parameters
aString

The string with which to compare the receiver.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The result of invoking `compare:options:` (page 981) with `NSCaseInsensitiveSearch` as the only option.

Discussion
If you are comparing strings to present to the end-user, you should typically use `localizedCaseInsensitiveCompare:` (page 1013) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `localizedCaseInsensitiveCompare:` (page 1013)
- `compare:options:` (page 981)

Declared In
`NSString.h`

characterAtIndex:

Returns the character at a given array position.

```
- (unichar)characterAtIndex:(NSUInteger)index
```

Parameters

index

The index of the character to retrieve. The index value must not lie outside the bounds of the receiver.

Return Value

The character at the array position given by *index*.

Discussion

Raises an `NSRangeException` if *index* lies beyond the end of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `getCharacters:` (page 992)
- `getCharacters:range:` (page 992)

Declared In

`NSString.h`

commonPrefixWithString:options:

Returns a string containing prefix the receiver and a given string have in common.

```
- (NSString *)commonPrefixWithString:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

mask

Options for the comparison. The following search options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

A string containing characters the receiver and *aString* have in common, starting from the beginning of each up to the first characters that aren't equivalent.

Discussion

The returned string is based on the characters of the receiver. For example, if the receiver is "Ma¨dchen" and *aString* is "Mädchenschule", the string returned is "Ma¨dchen", not "Mädchen".

Availability

Available in iPhone OS 2.0 and later.

See Also

- `hasPrefix:` (page 997)

Declared In
NSString.h

compare:

Returns the result of invoking compare:options:range: (page 982) with no options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
The result of invoking compare:options:range: (page 982) with no options and the receiver's full extent as the range.

Discussion
If you are comparing strings to present to the end-user, you should typically use localizedCompare: (page 1013) or localizedCaseInsensitiveCompare: (page 1013) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- localizedCompare: (page 1013)
- localizedCaseInsensitiveCompare: (page 1013)
- compare:options: (page 981)
- caseInsensitiveCompare: (page 979)
- isEqualToString: (page 1010)

Declared In
NSString.h

compare:options:

Returns the result of invoking compare:options:range: (page 982) with a given mask as the options and the receiver's full extent as the range.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`. See *String Programming Guide for Cocoa* for details on these options.

Return Value

The result of invoking `compare:options:range:` (page 982) with a given mask as the options and the receiver's full extent as the range.

Discussion

If you are comparing strings to present to the end-user, you should typically use `localizedCompare:` (page 1013) or `localizedCaseInsensitiveCompare:` (page 1013) instead, or use `compare:options:range:locale:` (page 983) and pass the user's locale.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `localizedCompare:` (page 1013)
- `localizedCaseInsensitiveCompare:` (page 1013)
- `compare:options:range:locale:` (page 983)
- `caseInsensitiveCompare:` (page 979)
- `isEqualToString:` (page 1010)

Declared In

`NSString.h`

compare:options:range:

Returns the result of invoking `compare:options:range:locale:` (page 983) with a `nil` locale.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

Return Value

The result of invoking `compare:options:range:locale:` (page 983) with a `nil` locale.

Discussion

If you are comparing strings to present to the end-user, you should typically use `compare:options:range:locale:` (page 983) instead and pass the user's locale (`currentLocale` (page 537) [`NSLocale`]).

Availability

Available in iPhone OS 2.0 and later.

See Also

- `localizedCompare:` (page 1013)
- `localizedCaseInsensitiveCompare:` (page 1013)
- `compare:options:` (page 981)
- `caseInsensitiveCompare:` (page 979)
- `isEqualToString:` (page 1010)

Declared In

`NSString.h`

compare:options:range:locale:

Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.

```
- (NSComparisonResult)compare:(NSString *)aString
    options:(NSStringCompareOptions)mask range:(NSRange)range locale:(id)locale
```

Parameters

aString

The string with which to compare the range of the receiver specified by *range*.

This value must not be `nil`. If this value is `nil`, the behavior is undefined and may change in future versions of Mac OS X.

mask

Options for the search—you can combine any of the following using a C bitwise OR operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSNumericSearch`.

See *String Programming Guide for Cocoa* for details on these options.

range

The range of the receiver over which to perform the comparison. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if *range* exceeds the bounds of the receiver.

locale

An instance of `NSLocale`. If this value not `nil` and is not an instance of `NSLocale`, uses the current locale instead.

If you are comparing strings to present to the end-user, you should typically pass the user's locale (`currentLocale` (page 537) [`NSLocale`]).

Return Value
`NSOrderedAscending` if the substring of the receiver given by *`range`* precedes *`aString`* in lexical ordering for the locale given in *`dict`*, `NSOrderedSame` if the substring of the receiver and *`aString`* are equivalent in lexical value, and `NSOrderedDescending` if the substring of the receiver follows *`aString`*.

Special Considerations
Prior to Mac OS X v10.5, the *`locale`* argument was an instance of `NSDictionary`. On Mac OS X v10.5 and later, if you pass an instance of `NSDictionary` the current locale is used instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `localizedCompare:` (page 1013)
- `localizedCaseInsensitiveCompare:` (page 1013)
- `caseInsensitiveCompare:` (page 979)
- `compare:` (page 981)
- `compare:options:` (page 981)
- `compare:options:range:` (page 982)
- `isEqualToString:` (page 1010)

Declared In
`NSString.h`

completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:
Interprets the receiver as a path in the file system and attempts to perform filename completion, returning a numeric value that indicates whether a match was possible, and by reference the longest path that matches the receiver.

```
- (NSUInteger)completePathIntoString:(NSString **)outputName caseSensitive:(BOOL)flag
    matchesIntoArray:(NSArray **)outputArray filterTypes:(NSArray *)filterTypes
```

Parameters
`outputName`
Upon return, contains the longest path that matches the receiver.

`flag`
If `YES`, the methods considers case for possible completions.

`outputArray`
Upon return, contains all matching filenames.

`filterTypes`
An array of `NSString` objects specifying path extensions to consider for completion. only paths whose extensions (not including the extension separator) match one of those strings.

Return Value
`0` if no matches are found and `1` if exactly one match is found. In the case of multiple matches, returns the actual number of matching paths if *`outputArray`* is provided, or simply a positive value if *`outputArray`* is `NULL`.

Discussion
You can check for the existence of matches without retrieving by passing `NULL` as *`outputArray`*.

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPathUtilities.h

componentsSeparatedByCharactersInSet:

Returns an array containing substrings from the receiver that have been divided by characters in a given set.

```
- (NSArray *)componentsSeparatedByCharactersInSet:(NSCharacterSet *)separator
```

Parameters

separator
A character set containing the characters to to use to split the receiver. Must not be nil.

Return Value
An NSArray object containing substrings from the receiver that have been divided by characters in *separator*.

Discussion
The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator characters produce empty strings in the result. Similarly, if the string begins or ends with separator characters, the first or last substring, respectively, is empty.

Availability
Available in iPhone OS 2.0 and later.

See Also
- componentsSeparatedByString: (page 985)
- stringByTrimmingCharactersInSet: (page 1037)

Declared In
NSString.h

componentsSeparatedByString:

Returns an array containing substrings from the receiver that have been divided by a given separator.

```
- (NSArray *)componentsSeparatedByString:(NSString *)separator
```

Parameters

separator
The separator string.

Return Value
An NSArray object containing substrings from the receiver that have been divided by *separator*.

Discussion

The substrings in the array appear in the order they did in the receiver. Adjacent occurrences of the separator string produce empty strings in the result. Similarly, if the string begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code fragment:

```
NSString *list = @"Norman, Stanley, Fletcher";
NSArray *listItems = [list componentsSeparatedByString:@", "];
```

produces an array `{ @"Norman", @"Stanley", @"Fletcher" }`.

If *`list`* begins with a comma and space—for example, `", Norman, Stanley, Fletcher"`—the array has these contents: `{ @"", @"Norman", @"Stanley", @"Fletcher" }`

If *`list`* has no separators—for example, `"Norman"`—the array contains the string itself, in this case `{ @"Norman" }`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`componentsJoinedByString:` (page 46) (`NSArray`)

`- pathComponents` (page 1016)

Declared In

`NSString.h`

cStringUsingEncoding:

Returns a representation of the receiver as a C string using a given encoding.

```
- (const char *)cStringUsingEncoding:(NSStringEncoding)encoding
```

Parameters

`encoding`

The encoding for the returned C string.

Return Value

A C string representation of the receiver using the encoding specified by *`encoding`*. Returns `NULL` if the receiver cannot be losslessly converted to *`encoding`*.

Discussion

The returned C string is guaranteed to be valid only until either the receiver is freed, or until the current autorelease pool is emptied, whichever occurs first. You should copy the C string or use `getCString:maxLength:encoding:` (page 993) if it needs to store the C string beyond this time.

You can use `canBeConvertedToEncoding:` (page 978) to check whether a string can be losslessly converted to *`encoding`*. If it can't, you can use `dataUsingEncoding:allowLossyConversion:` (page 987) to get a C-string representation using *`encoding`*, allowing some loss of information (note that the data returned by `dataUsingEncoding:allowLossyConversion:` is not a strict C-string since it does not have a `NULL` terminator).

Availability

Available in iPhone OS 2.0 and later.

See Also
- canBeConvertedToEncoding: (page 978)
+ defaultCStringEncoding (page 969)
- getCharacters: (page 992)
- UTF8String (page 1040)

Declared In
NSString.h

dataUsingEncoding:

Returns an NSData object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
A string encoding.

Return Value
The result of invoking dataUsingEncoding:allowLossyConversion: (page 987) with NO as the second argument (that is, requiring lossless conversion).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

dataUsingEncoding:allowLossyConversion:

Returns an NSData object containing a representation of the receiver encoded using a given encoding.

```
- (NSData *)dataUsingEncoding:(NSStringEncoding)encoding
    allowLossyConversion:(BOOL)flag
```

Parameters
encoding
A string encoding.

flag
If YES, then allows characters to be removed or altered in conversion.

Return Value
An NSData object containing a representation of the receiver encoded using *encoding*. Returns nil if *flag* is NO and the receiver can't be converted without losing some information (such as accents or case).

Discussion
If *flag* is YES and the receiver can't be converted without losing some information, some characters may be removed or altered in conversion. For example, in converting a character from NSUnicodeStringEncoding to NSASCIIStringEncoding, the character 'Á' becomes 'A', losing the accent.

This method creates an external representation (with a byte order marker, if necessary, to indicate endianness) to ensure that the resulting `NSData` object can be written out to a file safely. The result of this method, when lossless conversion is made, is the default "plain text" format for encoding and is the recommended way to save or transmit a string object.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ availableStringEncodings` (page 969)
`- canBeConvertedToEncoding:` (page 978)

Declared In
`NSString.h`

decomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form D.

```
- (NSString *)decomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form D.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- precomposedStringWithCanonicalMapping` (page 1017)
`- decomposedStringWithCompatibilityMapping` (page 988)

Declared In
`NSString.h`

decomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KD.

```
- (NSString *)decomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KD.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- precomposedStringWithCompatibilityMapping` (page 1018)
`- decomposedStringWithCanonicalMapping` (page 988)

Declared In
`NSString.h`

description

Returns the receiver.

```
- (NSString *)description
```

Return Value
The receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

doubleValue

Returns the floating-point value of the receiver's text as a `double`.

```
- (double)doubleValue
```

Return Value
The floating-point value of the receiver's text as a `double`. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, 0.0 on underflow. Returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method skips any whitespace at the beginning of the string. This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `floatValue` (page 990)
- `longLongValue` (page 1014)
- `integerValue` (page 1008)
- `scanDouble:` (page 906) (`NSScanner`)

Declared In
NSString.h

fastestEncoding

Returns the fastest encoding to which the receiver may be converted without loss of information.

```
- (NSStringEncoding)fastestEncoding
```

Return Value
The fastest encoding to which the receiver may be converted without loss of information.

Discussion
"Fastest" applies to retrieval of characters from the string. This encoding may not be space efficient.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `smallestEncoding` (page 1025)
- `getCharacters:range:` (page 992)

Declared In
`NSString.h`

fileSystemRepresentation

Returns a file system-specific representation of the receiver.

```
- (const char *)fileSystemRepresentation
```

Return Value
A file system-specific representation of the receiver, as described for `getFileSystemRepresentation:maxLength:` (page 993).

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the representation or use `getFileSystemRepresentation:maxLength:` (page 993) if it needs to store the representation outside of the autorelease context in which the representation is created.

Raises an `NSCharacterConversionException` if the receiver can't be represented in the file system's encoding.

Note that this method only works with file paths (not, for example, string representations of URLs).

To convert a `char *` path (such as you might get from a C library routine) to an `NSString` object, use `NSFileManager`'s `stringWithFileSystemRepresentation:length:` (page 411) method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSPathUtilities.h`

floatValue

Returns the floating-point value of the receiver's text as a `float`.

```
- (float)floatValue
```

Return Value
The floating-point value of the receiver's text as a `float`, skipping whitespace at the beginning of the string. Returns `HUGE_VAL` or `-HUGE_VAL` on overflow, `0.0` on underflow. Also returns `0.0` if the receiver doesn't begin with a valid text representation of a floating-point number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `doubleValue` (page 989)
- `longLongValue` (page 1014)
- `integerValue` (page 1008)
- `scanFloat:` (page 906) (`NSScanner`)

Declared In
`NSString.h`

getBytes:maxLength:usedLength:encoding:options:range:remainingRange:

Gets a given range of characters as bytes in a specified encoding.

```
- (BOOL)getBytes:(void *)buffer maxLength:(NSUInteger)maxBufferCount
    usedLength:(NSUInteger *)usedBufferCount encoding:(NSStringEncoding)encoding
    options:(NSStringEncodingConversionOptions)options range:(NSRange)range
    remainingRange:(NSRangePointer)leftover
```

Parameters
buffer
: A buffer into which to store the bytes from the receiver. The returned bytes are *not* `NULL`-terminated.

maxBufferCount
: The maximum number of bytes to write to *buffer*.

usedBufferCount
: The number of bytes used from *buffer*. Pass `NULL` if you do not need this value.

encoding
: The encoding to use for the returned bytes.

options
: A mask to specify options to use for converting the receiver's contents to *encoding* (if conversion is necessary).

range
: The range of characters in the receiver to get.

leftover
: The remaining range. Pass `NULL` If you do not need this value.

Return Value
`YES` if some characters were converted, otherwise `NO`.

Discussion
Conversion might stop when the buffer fills, but it might also stop when the conversion isn't possible due to the chosen encoding.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

getCharacters:

Returns by reference the characters from the receiver.

```
- (void)getCharacters:(unichar *)buffer
```

Parameters

buffer

Upon return, contains the characters from the receiver. *buffer* must be large enough to contain all characters in the string (`[string length]*sizeof(unichar)`).

Discussion

Invokes `getCharacters:range:` (page 992) with *buffer* and the entire extent of the receiver as the range.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- length` (page 1011)

Declared In

`NSString.h`

getCharacters:range:

Copies characters from a given range in the receiver into a given buffer.

```
- (void)getCharacters:(unichar *)buffer range:(NSRange)aRange
```

Parameters

buffer

Upon return, contains the characters from the receiver. *buffer* must be large enough to contain the characters in the range *aRange* (`aRange.length*sizeof(unichar)`).

aRange

The range of characters to retrieve. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the bounds of the receiver.

Discussion

This method does not add a `NULL` character.

The abstract implementation of this method uses `characterAtIndex:` (page 980) repeatedly, correctly extracting the characters, though very inefficiently. Subclasses should override it to provide a fast implementation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

getCString:maxLength:encoding:

Converts the receiver's content to a given encoding and stores them in a buffer.

```
- (BOOL)getCString:(char *)buffer maxLength:(NSUInteger)maxBufferCount
    encoding:(NSStringEncoding)encoding
```

Parameters

buffer

Upon return, contains the converted C-string plus the NULL termination byte. The buffer must include room for *maxBufferCount* bytes.

maxBufferCount

The maximum number of bytes in the string to return in buffer (*including* the NULL termination byte).

encoding

The encoding for the returned C string.

Return Value

YES if the operation was successful, otherwise NO. Returns NO if conversion is not possible due to encoding errors or if *buffer* is too small.

Discussion

Note that in the treatment of the *maxBufferCount* argument, this method differs from the deprecated getCString:maxLength: method which it replaces. (The buffer should include room for *maxBufferCount* bytes; this number should accomodate the expected size of the return value plus the NULL termination byte, which this method adds.)

You can use canBeConvertedToEncoding: (page 978) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use dataUsingEncoding:allowLossyConversion: (page 987) to get a C-string representation using *encoding*, allowing some loss of information (note that the data returned by dataUsingEncoding:allowLossyConversion: is not a strict C-string since it does not have a NULL terminator).

Availability

Available in iPhone OS 2.0 and later.

See Also

- cStringUsingEncoding: (page 986)
- canBeConvertedToEncoding: (page 978)
- getCharacters: (page 992)
- UTF8String (page 1040)

Declared In

NSString.h

getFileSystemRepresentation:maxLength:

Interprets the receiver as a system-independent path and fills a buffer with a C-string in a format and encoding suitable for use with file-system calls.

```
- (BOOL)getFileSystemRepresentation:(char *)buffer maxLength:(NSUInteger)maxLength
```

Parameters

buffer

Upon return, contains a C-string that represent the receiver as as a system-independent path, plus the NULL termination byte. The size of *buffer* must be large enough to contain *maxLength* bytes.

maxLength

The maximum number of bytes in the string to return in *buffer* (including a terminating NULL character, which this method adds).

Return Value

YES if *buffer* is successfully filled with a file-system representation, otherwise NO (for example, if *maxLength* would be exceeded or if the receiver can't be represented in the file system's encoding).

Discussion

This method operates by replacing the abstract path and extension separator characters ('/' and '.' respectively) with their equivalents for the operating system. If the system-specific path or extension separator appears in the abstract representation, the characters it is converted to depend on the system (unless they're identical to the abstract separators).

Note that this method only works with file paths (not, for example, string representations of URLs).

The following example illustrates the use of the *maxLength* argument. The first method invocation returns failure as the file representation of the string (@"/mach_kernel") is 12 bytes long and the value passed as the *maxLength* argument (12) does not allow for the addition of a NULL termination byte.

```
char filenameBuffer[13];
BOOL success;
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:12];
// success == NO
// Changing the length to include the NULL character does work
success = [@"/mach_kernel" getFileSystemRepresentation:filenameBuffer
maxLength:13];
// success == YES
```

Availability

Available in iPhone OS 2.0 and later.

See Also

- fileSystemRepresentation (page 990)

Declared In

NSPathUtilities.h

getLineStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first line and the end of the last line touched by the given range.

```
- (void)getLineStart:(NSUInteger *)startIndex end:(NSUInteger *)lineEndIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex

Upon return, contains the index of the first character of the line containing the beginning of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

lineEndIndex

Upon return, contains the index of the first character past the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex

Upon return, contains the index of the first character of the terminator of the line containing the end of *aRange*. Pass NULL if you do not need this value (in which case the work to compute the value isn't performed).

aRange

A range within the receiver. The value must not exceed the bounds of the receiver.

Important: Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Discussion

A line is delimited by any of these characters, the longest possible sequence being preferred to any shorter:

- U+000D (\r or CR)
- U+2028 (Unicode line separator)
- U+000A (\n or LF)
- U+2029 (Unicode paragraph separator)
- \r\n, in that order (also known as CRLF)

If *aRange* is contained with a single line, of course, the returned indexes all belong to that line. You can use the results of this method to construct ranges for lines by using the start index as the range's location and the difference between the end index and the start index as the range's length.

Availability

Available in iPhone OS 2.0 and later.

See Also

- lineRangeForRange: (page 1012)
- substringWithRange: (page 1039)

Declared In

NSString.h

getParagraphStart:end:contentsEnd:forRange:

Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.

```
- (void)getParagraphStart:(NSUInteger *)startIndex end:(NSUInteger *)endIndex
    contentsEnd:(NSUInteger *)contentsEndIndex forRange:(NSRange)aRange
```

Parameters

startIndex

Upon return, contains the index of the first character of the paragraph containing the beginning of *aRange*. Pass `NULL` if you do not need this value (in which case the work to compute the value isn't performed).

endIndex

Upon return, contains the index of the first character past the terminator of the paragraph containing the end of *aRange*. Pass `NULL` if you do not need this value (in which case the work to compute the value isn't performed).

contentsEndIndex

Upon return, contains the index of the first character of the terminator of the paragraph containing the end of *aRange*. Pass `NULL` if you do not need this value (in which case the work to compute the value isn't performed).

aRange

A range within the receiver. The value must not exceed the bounds of the receiver.

Discussion

If *aRange* is contained with a single paragraph, of course, the returned indexes all belong to that paragraph. Similar to `getLineStart:end:contentsEnd:forRange:` (page 994), you can use the results of this method to construct the ranges for paragraphs.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- paragraphRangeForRange:` (page 1015)

Declared In

`NSString.h`

hash

Returns an unsigned integer that can be used as a hash table address.

```
- (NSUInteger)hash
```

Return Value

An unsigned integer that can be used as a hash table address.

Discussion

If two string objects are equal (as determined by the `isEqualToString:` (page 1010) method), they must have the same hash value. The abstract implementation of this method fulfills this requirement, so subclasses of `NSString` shouldn't override it.

You should not rely on this method returning the same hash value across releases of Mac OS X.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSString.h

hasPrefix:

Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.

```
- (BOOL)hasPrefix:(NSString *)aString
```

Parameters

aString
A string.

Return Value
YES if *aString* matches the beginning characters of the receiver, otherwise NO. Returns NO if *aString* is empty.

Discussion
This method is a convenience for comparing strings using the NSAnchoredSearch option. See *String Programming Guide for Cocoa* for more information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- hasSuffix: (page 997)
- compare:options:range: (page 982)

Declared In
NSString.h

hasSuffix:

Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.

```
- (BOOL)hasSuffix:(NSString *)aString
```

Parameters

aString
A string.

Return Value
YES if *aString* matches the ending characters of the receiver, otherwise NO. Returns NO if *aString* is empty.

Discussion
This method is a convenience for comparing strings using the NSAnchoredSearch and NSBackwardsSearch options. See *String Programming Guide for Cocoa* for more information.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `hasPrefix:` (page 997)
- `compare:options:range:` (page 982)

Declared In
`NSString.h`

init

Returns an initialized `NSString` object that contains no characters.

```
- (id)init
```

Return Value
An initialized `NSString` object that contains no characters. The returned object may be different from the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ string` (page 972)

Declared In
`NSString.h`

initWithBytes:length:encoding:

Returns an initialized `NSString` object containing a given number of bytes from a given C array of bytes in a given encoding.

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding
```

Parameters

bytes
A C array of bytes in the encoding specified by *encoding*. The array must not contain `NULL`.

length
The number of bytes to use from *bytes*.

encoding
The character encoding of *bytes*.

Return Value
An initialized `NSString` object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 999)

Declared In
NSString.h

initWithBytesNoCopy:length:encoding:freeWhenDone:

Returns an initialized NSString object that contains a given number of bytes from a given C array of bytes in a given encoding, and optionally frees the array on deallocation.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding freeWhenDone:(BOOL)flag
```

Parameters

bytes
A C array of bytes in the encoding specified by *encoding*. The array must not contain NULL.

length
The number of bytes to use from *bytes*.

encoding
The character encoding of *bytes*.

flag
If YES, the receiver will free the memory when it no longer needs the data; if NO it won't.

Return Value
An initialized NSString object containing *length* bytes from *bytes* interpreted using the encoding *encoding*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not freed even if *flag* is YES. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithBytes:length:encoding: (page 998)

Declared In
NSString.h

initWithCharacters:length:

Returns an initialized NSString object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharacters:(const unichar *)characters length:(NSUInteger)length
```

Parameters

characters

A C array of Unicode characters; the value must not be NULL.

Important: Raises an exception if *characters* is NULL, even if *length* is 0.

length

The number of characters to use from *characters*.

Return Value

An initialized NSString object containing *length* characters taken from *characters*. The returned object may be different from the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ stringWithCharacters:length: (page 972)

Declared In

NSString.h

initWithCharactersNoCopy:length:freeWhenDone:

Returns an initialized NSString object that contains a given number of characters from a given C array of Unicode characters.

```
- (id)initWithCharactersNoCopy:(unichar *)characters length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Parameters

characters

A C array of Unicode characters.

length

The number of characters to use from *characters*.

flag

If YES, the receiver will free the memory when it no longer needs the characters; if NO it won't.

Return Value

An initialized NSString object that contains *length* characters from *characters*. The returned object may be different from the original receiver.

Special Considerations

If an error occurs during the creation of the string, then *bytes* is not freed even if *flag* is YES. In this case, the caller is responsible for freeing the buffer. This allows the caller to continue trying to create a string with the buffer, without having the buffer deallocated.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ stringWithCharacters:length: (page 972)

Declared In
NSString.h

initWithContentsOfFile:encoding:error:

Returns an NSString object initialized by reading data from the file at a given path using a given encoding.

```
- (id)initWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

The encoding of the file at *path*.

error

If an error occurs, upon return contains an NSError object that describes the problem. If you are not interested in possible errors, pass in NULL.

Return Value
An NSString object initialized by reading data from the file named by *path* using the encoding, *enc*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns nil.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ stringWithContentsOfFile:encoding:error: (page 973)
- initWithContentsOfFile:usedEncoding:error: (page 1001)

Declared In
NSString.h

initWithContentsOfFile:usedEncoding:error:

Returns an NSString object initialized by reading data from the file at a given path and returns by reference the encoding used to interpret the characters.

```
- (id)initWithContentsOfFile:(NSString *)path usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

path

A path to a file.

enc

Upon return, if the file is read successfully, contains the encoding used to interpret the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

An `NSString` object initialized by reading data from the file named by *path*. The returned object may be different from the original receiver. If the file can't be opened or there is an encoding error, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithContentsOfFile:encoding:error:` (page 973)

`- initWithContentsOfFile:encoding:error:` (page 1001)

Declared In

`NSString.h`

initWithContentsOfURL:encoding:error:

Returns an `NSString` object initialized by reading data from a given URL interpreted using a given encoding.

```
- (id)initWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)enc
    error:(NSError **)error
```

Parameters

url

The URL to read.

enc

The encoding of the file at *path*.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

An `NSString` object initialized by reading data from *url*. The returned object may be different from the original receiver. If the URL can't be opened or there is an encoding error, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithContentsOfURL:encoding:error:` (page 974)

Declared In

`NSString.h`

initWithContentsOfURL:usedEncoding:error:

Returns an `NSString` object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

```
- (id)initWithContentsOfURL:(NSURL *)url usedEncoding:(NSStringEncoding *)enc
    error:(NSError **)error
```

Parameters

url

The URL from which to read data.

enc

Upon return, if *url* is read successfully, contains the encoding used to interpret the data.

error

If an error occurs, upon returns contains an `NSError` object that describes the problem. If you are not interested in possible errors, pass in `NULL`.

Return Value

An `NSString` object initialized by reading data from *url*. If *url* can't be opened or the encoding cannot be determined, returns `nil`. The returned initialized object might be different from the original receiver

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithContentsOfURL:usedEncoding:error:` (page 975)

Declared In

`NSString.h`

initWithCString:encoding:

Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.

```
- (id)initWithCString:(const char *)nullTerminatedCString
    encoding:(NSStringEncoding)encoding
```

Parameters

nullTerminatedCString

A C array of characters. The array must end with a `NULL` character; intermediate `NULL` characters are not allowed.

encoding

The encoding of *nullTerminatedCString*.

Return Value

An `NSString` object initialized using the characters from *nullTerminatedCString*. The returned object may be different from the original receiver

Discussion

If *nullTerminatedCString* is not a `NULL`-terminated C string, or *encoding* does not match the actual encoding, the results are undefined.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ defaultCStringEncoding` (page 969)

Declared In
NSString.h

initWithData:encoding:

Returns an NSString object initialized by converting given data into Unicode characters using a given encoding.

```
- (id)initWithData:(NSData *)data encoding:(NSStringEncoding)encoding
```

Parameters

data

An NSData object containing bytes in *encoding* and the default plain text format (that is, pure content with no attributes or other markups) for that encoding.

encoding

The encoding used by *data*.

Return Value
An NSString object initialized by converting the bytes in *data* into Unicode characters using *encoding*. The returned object may be different from the original receiver. Returns nil if the initialization fails for some reason (for example if *data* does not represent valid data for *encoding*).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

initWithFormat:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted.

```
- (id)initWithFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *format* is nil.

...

A comma-separated list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which the remaining argument values are substituted according to the canonical locale. The returned object may be different from the original receiver.

Discussion
Invokes `initWithFormat:locale:arguments:` (page 1006) with `nil` as the locale, hence using the canonical locale to format numbers. This is useful, for example, if you want to produce "non-localized" formatting which needs to be written out to files and parsed back later.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ stringWithFormat:` (page 976)
`- initWithFormat:locale:arguments:` (page 1006)

Declared In
`NSString.h`

initWithFormat:arguments:
Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

```
- (id)initWithFormat:(NSString *)format arguments:(va_list)argList
```

Parameters
format
A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

argList
A list of arguments to substitute into *format*.

Return Value
An *NSString* object initialized by using *format* as a template into which the values in *argList* are substituted according to the user's default locale. The returned object may be different from the original receiver.

Discussion
Invokes `initWithFormat:locale:arguments:` (page 1006) with `nil` as the locale.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ stringWithFormat:` (page 976)

Declared In
`NSString.h`

initWithFormat:locale:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale ...
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

locale

This may be an instance of `NSDictionary` containing locale information or an instance of `NSLocale`. If this value is `nil`, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

...

A comma-separated list of arguments to substitute into *format*.

Discussion

Invokes `initWithFormat:locale:arguments:` (page 1006) with *locale* as the locale.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ localizedStringWithFormat:` (page 970)

Declared In

`NSString.h`

initWithFormat:locale:arguments:

Returns an *NSString* object initialized by using a given format string as a template into which the remaining argument values are substituted according to given locale information.

```
- (id)initWithFormat:(NSString *)format locale:(id)locale arguments:(va_list)argList
```

Parameters

format

A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

locale

This may be an instance of `NSDictionary` containing locale information or an instance of `NSLocale`. If this value is `nil`, uses the canonical locale.

To use a dictionary containing the current user's locale, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

argList

A list of arguments to substitute into *format*.

Return Value

An *NSString* object initialized by using *format* as a template into which values in *argList* are substituted according the locale information in *locale*. The returned object may be different from the original receiver.

Discussion

The following code fragment illustrates how to create a string from *myArgs*, which is derived from a string object with the value "Cost:" and an `int` with the value 32:

```
va_list myArgs;

NSString *myString = [[NSString alloc] initWithFormat:@"%@:  %d\n"
        locale:[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]
        arguments:myArgs];
```

The resulting string has the value "`Cost: 32\n`".

See *String Programming Guide for Cocoa* for more information.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithFormat:arguments:` (page 1005)

Declared In

`NSString.h`

initWithString:

Returns an `NSString` object initialized by copying the characters from another given string.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString

The string from which to copy characters. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

Return Value

An `NSString` object initialized by copying the characters from *aString*. The returned object may be different from the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithString:` (page 976)

Declared In

`NSString.h`

initWithUTF8String:

Returns an `NSString` object initialized by copying the characters a given C array of UTF8-encoded bytes.

```
- (id)initWithUTF8String:(const char *)bytes
```

Parameters

bytes

A `NULL`-terminated C array of bytes in UTF-8 encoding. This value must not be `NULL`.

Important: Raises an exception if *bytes* is `NULL`.

Return Value

An `NSString` object initialized by copying the bytes from *bytes*. The returned object may be different from the original receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ stringWithUTF8String:` (page 977)

Declared In

`NSString.h`

integerValue

Returns the `NSInteger` value of the receiver's text.

```
- (NSInteger)integerValue
```

Return Value
The `NSInteger` value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `doubleValue` (page 989)
- `floatValue` (page 990)
- `scanInt:` (page 907) (`NSScanner`)

Declared In
`NSString.h`

intValue
Returns the integer value of the receiver's text.

```
- (int)intValue
```

Return Value
The integer value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns `INT_MAX` or `INT_MIN` on overflow. Returns `0` if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an `NSScanner` object for localized scanning of numeric values from a string.

Special Considerations
On Mac OS X v10.5 and later, use `integerValue` (page 1008) instead.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `integerValue` (page 1008)
- `doubleValue` (page 989)
- `floatValue` (page 990)
- `scanInt:` (page 907) (`NSScanner`)

Declared In
`NSString.h`

isAbsolutePath

Returning a Boolean value that indicates whether the receiver represents an absolute path.

```
- (BOOL)isAbsolutePath
```

Return Value

`YES` if the receiver (if interpreted as a path) represents an absolute path, otherwise `NO` (if the receiver represents a relative path).

Discussion

See *String Programming Guide for Cocoa* for more information on paths.

Note that this method only works with file paths (not, for example, string representations of URLs). The method does not check the filesystem for the existence of the path (use `fileExistsAtPath:` (page 402) or similar methods in `NSFileManager` for that task).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPathUtilities.h`

isEqualToString:

Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.

```
- (BOOL)isEqualToString:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

Return Value

`YES` if *aString* is equivalent to the receiver (if they have the same `id` or if they are `NSOrderedSame` in a literal comparison), otherwise `NO`.

Discussion

The comparison uses the canonical representation of strings, which for a particular string is the length of the string plus the Unicode characters that make up the string. When this method compares two strings, if the individual Unicodes are the same, then the strings are equal, regardless of the backing store. “Literal” when applied to string comparison means that various Unicode decomposition rules are not applied and Unicode characters are individually compared. So, for instance, “Ö” represented as the composed character sequence “O” and umlaut would not compare equal to “Ö” represented as one Unicode character.

Special Considerations

When you know both objects are strings, this method is a faster way to check equality than `isEqual:` (page 1306).

Availability

Available in iPhone OS 2.0 and later.

See Also
- compare:options:range: (page 982)

Declared In
NSString.h

lastPathComponent

Returns the last path component of the receiver.

```
- (NSString *)lastPathComponent
```

Return Value
The last path component of the receiver.

Discussion
The following table illustrates the effect of `lastPathComponent` on a variety of different paths:

Receiver's String Value	String Returned
"/tmp/scratch.tiff"	"scratch.tiff"
"/tmp/scratch"	"scratch"
"/tmp/"	"tmp"
"scratch"	"scratch"
"/"	"/"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPathUtilities.h

length

Returns the number of Unicode characters in the receiver.

```
- (NSUInteger)length
```

Return Value
The number of Unicode characters in the receiver.

Discussion
The number returned includes the individual characters of composed character sequences, so you cannot use this method to determine if a string will be visible when printed or how long it will appear.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `lengthOfBytesUsingEncoding:` (page 1012)
`sizeWithAttributes:` (`NSString` Additions)

Declared In
`NSString.h`

lengthOfBytesUsingEncoding:

Returns the number of bytes required to store the receiver in a given encoding.

```
- (NSUInteger)lengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters

enc
The encoding for which to determine the receiver's length.

Return Value
The number of bytes required to store the receiver in the encoding *enc* in a non-external representation. The length does not include space for a terminating `NULL` character.

Discussion
The result is exact and is returned in `O(n)` time.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `maximumLengthOfBytesUsingEncoding:` (page 1015)
- `length` (page 1011)

Declared In
`NSString.h`

lineRangeForRange:

Returns the range of characters representing the line or lines containing a given range.

```
- (NSRange)lineRangeForRange:(NSRange)aRange
```

Parameters

aRange
A range within the receiver.

Return Value
The range of characters representing the line or lines containing *aRange*, including the line termination characters.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `paragraphRangeForRange:` (page 1015)
- `getLineStart:end:contentsEnd:forRange:` (page 994)

- substringWithRange: (page 1039)

Declared In
NSString.h

localizedCaseInsensitiveCompare:

Returns an NSComparisonResult value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.

```
- (NSComparisonResult)localizedCaseInsensitiveCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
NSOrderedAscending the receiver precedes *aString* in lexical ordering, NSOrderedSame the receiver and *aString* are equivalent in lexical value, and NSOrderedDescending if the receiver follows *aString*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- compare:options:range:locale: (page 983)

Declared In
NSString.h

localizedCompare:

Returns an NSComparisonResult value that indicates the lexical ordering of the receiver and another given string using a localized comparison.

```
- (NSComparisonResult)localizedCompare:(NSString *)aString
```

Parameters

aString

The string with which to compare the receiver.

This value must not be nil. If this value is nil, the behavior is undefined and may change in future versions of Mac OS X.

Return Value
NSOrderedAscending the receiver precedes *string* in lexical ordering, NSOrderedSame the receiver and *string* are equivalent in lexical value, and NSOrderedDescending if the receiver follows *string*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- compare:options:range:locale: (page 983)

Declared In
NSString.h

longLongValue

Returns the long long value of the receiver's text.

```
- (long long)longLongValue
```

Return Value
The long long value of the receiver's text, assuming a decimal representation and skipping whitespace at the beginning of the string. Returns INT_MAX or INT_MIN on overflow. Returns 0 if the receiver doesn't begin with a valid decimal text representation of a number.

Discussion
This method uses formatting information stored in the non-localized value; use an NSScanner object for localized scanning of numeric values from a string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- doubleValue (page 989)
- floatValue (page 990)
- scanInt: (page 907) (NSScanner)

Declared In
NSString.h

lowercaseString

Returns lowercased representation of the receiver.

```
- (NSString *)lowercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding lowercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. The result of this statement:

```
lcString = [myString lowercaseString];
```

might not be equal to this statement:

```
lcString = [[myString uppercaseString] lowercaseString];
```

For example, the uppercase form of "ß" in German is "SS", so converting "Straße" to uppercase, then lowercase, produces this sequence of strings:

"Straße"
"STRASSE"

"strasse"

Availability
Available in iPhone OS 2.0 and later.

See Also
- `capitalizedString` (page 978)
- `uppercaseString` (page 1039)

Declared In
`NSString.h`

maximumLengthOfBytesUsingEncoding:

Returns the maximum number of bytes needed to store the receiver in a given encoding.

```
- (NSUInteger)maximumLengthOfBytesUsingEncoding:(NSStringEncoding)enc
```

Parameters

enc

The encoding for which to determine the receiver's length.

Return Value
The maximum number of bytes needed to store the receiver in *encoding* in a non-external representation. The length does not include space for a terminating `NULL` character.

Discussion
The result is an estimate and is returned in `O(1)` time; the estimate may be considerably greater than the actual length needed.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `lengthOfBytesUsingEncoding:` (page 1012)
- `length` (page 1011)

Declared In
`NSString.h`

paragraphRangeForRange:

Returns the range of characters representing the paragraph or paragraphs containing a given range.

```
- (NSRange)paragraphRangeForRange:(NSRange)aRange
```

Parameters

aRange

A range within the receiver. The range must not exceed the bounds of the receiver.

Return Value
The range of characters representing the paragraph or paragraphs containing *aRange*, including the paragraph termination characters.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `getParagraphStart:end:contentsEnd:forRange:` (page 995)
- `lineRangeForRange:` (page 1012)

Declared In
`NSString.h`

pathComponents

Returns an array of `NSString` objects containing, in order, each path component of the receiver.

```
- (NSArray *)pathComponents
```

Return Value
An array of `NSString` objects containing, in order, each path component of the receiver.

Discussion
The strings in the array appear in the order they did in the receiver. If the string begins or ends with the path separator, then the first or last component, respectively, will contain the separator. Empty components (caused by consecutive path separators) are deleted. For example, this code excerpt:

```
NSString *path = @"tmp/scratch";
NSArray *pathComponents = [path pathComponents];
```

produces an array with these contents:

Index	Path Component
0	“tmp”
1	“scratch”

If the receiver begins with a slash—for example, “`/tmp/scratch`”—the array has these contents:

Index	Path Component
0	“/”
1	“tmp”
2	“scratch”

If the receiver has no separators—for example, “`scratch`”—the array contains the string itself, in this case “`scratch`”.

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
+ pathWithComponents: (page 971)
- stringByStandardizingPath (page 1036)
- componentsSeparatedByString: (page 985)

Declared In
NSPathUtilities.h

pathExtension

Interprets the receiver as a path and returns the receiver's extension, if any.

```
- (NSString *)pathExtension
```

Return Value
The receiver's extension, if any (not including the extension divider).

Discussion
The following table illustrates the effect of pathExtension on a variety of different paths:

Receiver's String Value	String Returned
"/tmp/scratch.tiff"	"tiff"
"/tmp/scratch"	"" (an empty string)
"/tmp/"	"" (an empty string)
"/tmp/scratch..tiff"	"tiff"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSPathUtilities.h

precomposedStringWithCanonicalMapping

Returns a string made by normalizing the receiver's contents using Form C.

```
- (NSString *)precomposedStringWithCanonicalMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form C.

Availability
Available in iPhone OS 2.0 and later.

See Also
- precomposedStringWithCompatibilityMapping (page 1018)

- decomposedStringWithCanonicalMapping (page 988)

Declared In
NSString.h

precomposedStringWithCompatibilityMapping

Returns a string made by normalizing the receiver's contents using Form KC.

```
- (NSString *)precomposedStringWithCompatibilityMapping
```

Return Value
A string made by normalizing the receiver's contents using the Unicode Normalization Form KC.

Availability
Available in iPhone OS 2.0 and later.

See Also
- precomposedStringWithCanonicalMapping (page 1017)
- decomposedStringWithCompatibilityMapping (page 988)

Declared In
NSString.h

propertyList

Parses the receiver as a text representation of a property list, returning an NSString, NSData, NSArray, or NSDictionary object, according to the topmost element.

```
- (id)propertyList
```

Return Value
A property list representation of returning an NSString, NSData, NSArray, or NSDictionary object, according to the topmost element.

Discussion
The receiver must contain a string in a property list format. For a discussion of property list formats, see *Property List Programming Guide for Cocoa*.

Important: Raises an NSParseErrorException if the receiver cannot be parsed as a property list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- propertyListFromStringsFileFormat (page 1019)

Declared In
NSString.h

propertyListFromStringsFileFormat

Returns a dictionary object initialized with the keys and values found in the receiver.

```
- (NSDictionary *)propertyListFromStringsFileFormat
```

Return Value

A dictionary object initialized with the keys and values found in the receiver

Discussion

The receiver must contain text in the format used for `.strings` files. In this format, keys and values are separated by an equal sign, and each key-value pair is terminated with a semicolon. The value is optional—if not present, the equal sign is also omitted. The keys and values themselves are always strings enclosed in straight quotation marks. Comments may be included, delimited by `/*` and `*/` as for ANSI C comments. Here's a short example of a strings file:

```
/* Question in confirmation panel for quitting. */
"Confirm Quit" = "Are you sure you want to quit?";

/* Message when user tries to close unsaved document */
"Close or Save" = "Save changes before closing?";

/* Word for Cancel */
"Cancel";
```

Availability

Available in iPhone OS 2.0 and later.

See Also

`- propertyList` (page 1018)

Declared In

`NSString.h`

rangeOfCharacterFromSet:

Returns the range in the receiver of the first character found from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

Return Value

The range in the receiver of the first character found from *aSet*. Returns a range of `{NSNotFound, 0}`if none of the characters in *aSet* are found.

Discussion

Invokes `rangeOfCharacterFromSet:options:` (page 1020) with no options.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSString.h

rangeOfCharacterFromSet:options:

Returns the range in the receiver of the first character found, using given options, from a given character set.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask
```

Parameters

aSet

A character set. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aSet* is nil.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise OR operator: NSCaseInsensitiveSearch, NSLiteralSearch, NSBackwardsSearch. See *String Programming Guide for Cocoa* for details on these options.

Return Value
The range in the receiver of the first character found from *aSet*. Returns a range of {NSNotFound, 0} if none of the characters in *aSet* are found.

Discussion
Invokes rangeOfCharacterFromSet:options:range: (page 1020) with *mask* for the options and the entire extent of the receiver for the range.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

rangeOfCharacterFromSet:options:range:

Returns the range in the receiver of the first character found from a given character set found in a given range with given options.

```
- (NSRange)rangeOfCharacterFromSet:(NSCharacterSet *)aSet
    options:(NSStringCompareOptions)mask range:(NSRange)aRange
```

Parameters

aSet

A character set. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aSet* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range in which to search. *aRange* must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value

The range in the receiver of the first character found from *aSet* within *aRange*. Returns a range of `{NSNotFound, 0}` if none of the characters in *aSet* are found.

Discussion

Because pre-composed characters in *aSet* can match composed character sequences in the receiver, the length of the returned range can be greater than `1`. For example, if you search for "ü" in the string "`stru¨del`", the returned range is `{3,2}`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

rangeOfComposedCharacterSequenceAtIndex:

Returns the range in the receiver of the composed character sequence located at a given index.

```
- (NSRange)rangeOfComposedCharacterSequenceAtIndex:(NSUInteger)anIndex
```

Parameters

anIndex

The index of a character in the receiver. The value must not exceed the bounds of the receiver.

Return Value

The range in the receiver of the composed character sequence located at *anIndex*.

Discussion

The composed character sequence includes the first base character found at or before *anIndex*, and its length includes the base character and all non-base characters following the base character.

If you want to write a method to adjust an arbitrary range so it includes the composed character sequences on its boundaries, you can create a method such as the following:

```
- (NSRange)adjustRange:(NSRange)aRange
{
```

```
    NSUInteger index, endIndex;
    NSRange newRange, endRange;

    // Check for validity of range
    if ( aRange.location >= [self length] ||
         aRange.location + aRange.length > [self length] )
    {
        [NSException raise:NSRangeException format:@"Invalid  range %@.",
        NSStringFromRange(aRange)];
    }

    index = aRange.location;
    newRange = [self rangeOfComposedCharacterSequenceAtIndex:index];

    index = aRange.location + aRange.length - 1;
    endRange = [self rangeOfComposedCharacterSequenceAtIndex:index];
    endIndex = endRange.location + endRange.length;

    newRange.length = endIndex - newRange.location;

    return newRange;
}
```

First, `adjustRange:` corrects the location for the beginning of *aRange*, storing it in *newRange*. It then works at the end of *aRange*, correcting the location and storing it in *endIndex*. Finally, it sets the length of *newRange* to the difference between *endIndex* and the new range's location.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `rangeOfComposedCharacterSequencesForRange:` (page 1022)

Declared In
`NSString.h`

rangeOfComposedCharacterSequencesForRange:

Returns the range in the receiver of the composed character sequence in a given range.

```
- (NSRange)rangeOfComposedCharacterSequencesForRange:(NSRange)range
```

Parameters
range

A range in the receiver. The range must not exceed the bounds of the receiver.

Return Value
The range in the receiver of the composed character sequence in *range*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `rangeOfComposedCharacterSequenceAtIndex:` (page 1021)

Declared In
NSString.h

rangeOfString:

Returns the range of the first occurrence within the receiver of a given string.

```
- (NSRange)rangeOfString:(NSString *)aString
```

Parameters

aString

The string to search for. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

Return Value
An NSRange structure giving the location and length in the receiver of the first occurrence of *aString*. Returns {NSNotFound, 0} if *subString* is not found or is empty (@"").

Discussion
Invokes rangeOfString:options: (page 1023) with no options.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

rangeOfString:options:

Returns the range of the first occurrence within the receiver of a given string, subject to given options

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
```

Parameters

aString

The string to search for. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise OR operator: NSCaseInsensitiveSearch, NSLiteralSearch, NSBackwardsSearch, NSAnchoredSearch. See *String Programming Guide for Cocoa* for details on these options.

Return Value
An NSRange structure giving the location and length in the receiver of the first occurrence of *subString*, modulo the options in *mask*. Returns {NSNotFound, 0} if *subString* is not found or is empty (@"").

Discussion
Invokes `rangeOfString:options:range:` (page 1024) with the options specified by *mask* and the entire extent of the receiver as the range.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

rangeOfString:options:range:
Returns the range of the first occurrence within a given range of the receiver of a given string, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)subString options:(NSStringCompareOptions)mask
    range:(NSRange)aRange
```

Parameters
subString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *subString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

Return Value
An `NSRange` structure giving the location and length in the receiver of *subString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *subString* is not found or is empty (`@""`).

Discussion
The length of the returned range and that of *subString* may differ if equivalent composed character sequences are matched.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

rangeOfString:options:range:locale:

Returns the range of the first occurrence within a given range of the receiver of a given string, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)searchRange locale:(NSLocale *)locale
```

Parameters

subString

The string for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *aString* is `nil`.

mask

A mask specifying search options. The following options may be specified by combining them with the C bitwise `OR` operator: `NSCaseInsensitiveSearch`, `NSLiteralSearch`, `NSBackwardsSearch`, and `NSAnchoredSearch`. See *String Programming Guide for Cocoa* for details on these options.

aRange

The range within the receiver for which to search for *subString*.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the string.

locale

The locale to use when comparing the receiver with *subString*. If this value is `nil`, uses the current locale.

Return Value

An `NSRange` structure giving the location and length in the receiver of *subString* within *aRange* in the receiver, modulo the options in *mask*. The range returned is relative to the start of the string, not to the passed-in range. Returns `{NSNotFound, 0}` if *subString* is not found or is empty (`@""`).

Discussion

The length of the returned range and that of *subString* may differ if equivalent composed character sequences are matched.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

smallestEncoding

Returns the smallest encoding to which the receiver can be converted without loss of information.

```
- (NSStringEncoding)smallestEncoding
```

Return Value

The smallest encoding to which the receiver can be converted without loss of information.

Discussion
The returned encoding may not be the fastest for accessing characters, but is space-efficient. This method may take some time to execute.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `fastestEncoding` (page 989)
- `getCharacters:range:` (page 992)

Declared In
`NSString.h`

stringByAbbreviatingWithTildeInPath

Returns a new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory.

```
- (NSString *)stringByAbbreviatingWithTildeInPath
```

Return Value
A new string representing the receiver as a path with a tilde (~) substituted for the full path to the current user's home directory. Returns a new string matching the receiver if the receiver doesn't begin with a user's home directory.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1031)

Declared In
`NSPathUtilities.h`

stringByAddingPercentEscapesUsingEncoding:

Returns a representation of the receiver using a given encoding to determine the percent escapes necessary to convert the receiver into a legal URL string.

```
- (NSString *)stringByAddingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters
encoding
The encoding to use for the returned string.

Return Value
A representation of the receiver using *encoding* to determine the percent escapes necessary to convert the receiver into a legal URL string. Returns `nil` if *encoding* cannot encode a particular character

Discussion
See `CFURLCreateStringByAddingPercentEscapes` for more complex transformations.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stringByReplacingPercentEscapesUsingEncoding:` (page 1035)

Declared In
`NSURL.h`

stringByAppendingFormat:

Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.

```
- (NSString *)stringByAppendingFormat:(NSString *)format ...
```

Parameters

format

A format string. See Formatting String Objects for more information. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` if *format* is `nil`.

`...`

A comma-separated list of arguments to substitute into *format*.

Return Value
A string made by appending to the receiver a string constructed from *format* and the following arguments, in the manner of `stringWithFormat:` (page 976).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stringByAppendingString:` (page 1029)

Declared In
`NSString.h`

stringByAppendingPathComponent:

Returns a new string made by appending to the receiver a given string.

```
- (NSString *)stringByAppendingPathComponent:(NSString *)aString
```

Parameters

aString

The path component to append to the receiver.

Return Value
A new string made by appending *aString* to the receiver, preceded if necessary by a path separator.

Discussion
The following table illustrates the effect of this method on a variety of different paths, assuming that *aString* is supplied as "scratch.tiff":

Receiver's String Value	Resulting String
"/tmp"	"/tmp/scratch.tiff"
"/tmp/"	"/tmp/scratch.tiff"
"/"	"/scratch.tiff"
"" (an empty string)	"scratch.tiff"

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- stringsByAppendingPaths: (page 1037)
- stringByAppendingPathExtension: (page 1028)
- stringByDeletingLastPathComponent (page 1030)

Declared In
NSPathUtilities.h

stringByAppendingPathExtension:

Returns a new string made by appending to the receiver an extension separator followed by a given extension.

```
- (NSString *)stringByAppendingPathExtension:(NSString *)ext
```

Parameters
ext
The extension to append to the receiver.

Return Value
A new string made by appending to the receiver an extension separator followed by *ext*.

Discussion
The following table illustrates the effect of this method on a variety of different paths, assuming that *ext* is supplied as @"tiff":

Receiver's String Value	Resulting String
"/tmp/scratch.old"	"/tmp/scratch.old.tiff"
"/tmp/scratch."	"/tmp/scratch..tiff"

Receiver's String Value	Resulting String
"/tmp/"	"/tmp.tiff"
"scratch"	"scratch.tiff"

Note that adding an extension to @"/tmp/" causes the result to be @"/tmp.tiff" instead of @"/tmp/.tiff". This difference is because a file named @".tiff" is not considered to have an extension, so the string is appended to the last nonempty path component.

This method does not allow you to append file extensions to filenames starting with the tilde character (~).

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- stringByAppendingPathComponent: (page 1027)
- stringByDeletingPathExtension (page 1030)

Declared In
NSPathUtilities.h

stringByAppendingString:
Returns a new string made by appending a given string to the receiver.

```
- (NSString *)stringByAppendingString:(NSString *)aString
```

Parameters
aString
The string to append to the receiver. This value must not be nil.

Important: Raises an NSInvalidArgumentException if *aString* is nil.

Return Value
A new string made by appending *aString* to the receiver.

Discussion
This code excerpt, for example:

```
NSString *errorTag = @"Error: ";
NSString *errorString = @"premature end of file.";
NSString *errorMessage = [errorTag stringByAppendingString:errorString];
```

produces the string "Error: premature end of file.".

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByAppendingFormat:` (page 1027)

Declared In
`NSString.h`

stringByDeletingLastPathComponent

Returns a new string made by deleting the last path component from the receiver, along with any final path separator.

```
- (NSString *)stringByDeletingLastPathComponent
```

Return Value
A new string made by deleting the last path component from the receiver, along with any final path separator. If the receiver represents the root path it is returned unaltered.

Discussion
The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp"
"/tmp/lock/"	"/tmp"
"/tmp/"	"/"
"/tmp"	"/"
"/"	"/"
"scratch.tiff"	"" (an empty string)

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByDeletingPathExtension` (page 1030)
- `stringByAppendingPathComponent:` (page 1027)

Declared In
`NSPathUtilities.h`

stringByDeletingPathExtension

Returns a new string made by deleting the extension (if any, and only the last) from the receiver.

```
- (NSString *)stringByDeletingPathExtension
```

Return Value
a new string made by deleting the extension (if any, and only the last) from the receiver. Strips any trailing path separator before checking for an extension. If the receiver represents the root path, it is returned unaltered.

Discussion
The following table illustrates the effect of this method on a variety of different paths:

Receiver's String Value	Resulting String
"/tmp/scratch.tiff"	"/tmp/scratch"
"/tmp/"	"/tmp"
"scratch.bundle/"	"scratch"
"scratch..tiff"	"scratch."
".tiff"	".tiff"
"/"	"/"

Note that attempting to delete an extension from `@".tiff"` causes the result to be `@".tiff"` instead of an empty string. This difference is because a file named `@".tiff"` is not considered to have an extension, so nothing is deleted. Note also that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `pathExtension` (page 1017)
- `stringByDeletingLastPathComponent` (page 1030)

Declared In
`NSPathUtilities.h`

stringByExpandingTildeInPath
Returns a new string made by expanding the initial component of the receiver to its full path value.

```
- (NSString *)stringByExpandingTildeInPath
```

Return Value
A new string made by expanding the initial component of the receiver, if it begins with "~" or "`~user`", to its full path value. Returns a new string matching the receiver if the receiver's initial component can't be expanded.

Discussion
Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- stringByAbbreviatingWithTildeInPath (page 1026)

Declared In
NSPathUtilities.h

stringByFoldingWithOptions:locale:

Returns a string with the given character folding options applied.

```
- (NSString *)stringByFoldingWithOptions:(NSStringCompareOptions)options
    locale:(NSLocale *)locale
```

Parameters

options
A mask of compare flags with a suffix InsensitiveSearch.

locale
The locale to use for the folding.

Return Value
A string with the character folding options applied.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

stringByPaddingToLength:withString:startingAtIndex:

Returns a new string formed from the receiver by either removing characters from the end, or by appending as many occurrences as necessary of a given pad string.

```
- (NSString *)stringByPaddingToLength:(NSUInteger)newLength withString:(NSString
    *)padString startingAtIndex:(NSUInteger)padIndex
```

Parameters

newLength
The new length for the receiver.

padString
The string with which to extend the receiver.

padIndex
The index in *padString* from which to start padding.

Return Value
A new string formed from the receiver by either removing characters from the end, or by appending as many occurrences of *padString* as necessary.

Discussion
Here are some examples of usage:

```
[@"abc" stringByPaddingToLength: 9 withString: @"." startingAtIndex:0];
    // Results in "abc......"
```

```
[@"abc" stringByPaddingToLength: 2 withString: @"." startingAtIndex:0];
    // Results in "ab"

[@"abc" stringByPaddingToLength: 9 withString: @". " startingAtIndex:1];
    // Results in "abc . . ."
    // Notice that the first character in the padding is " "
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

stringByReplacingCharactersInRange:withString:

Returns a new string in which the characters in a specified range of the receiver are replaced by a given string.

```
- (NSString *)stringByReplacingCharactersInRange:(NSRange)range withString:(NSString
    *)replacement
```

Parameters

range
A range of characters in the receiver.

replacement
The string with which to replace the characters in *range*.

Return Value
A new string in which the characters in *range* of the receiver are replaced by *replacement*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- stringByReplacingOccurrencesOfString:withString: (page 1033)
- stringByReplacingOccurrencesOfString:withString:options:range: (page 1034)
- stringByReplacingPercentEscapesUsingEncoding: (page 1035)

Declared In
NSString.h

stringByReplacingOccurrencesOfString:withString:

Returns a new string in which all occurrences of a target string in the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement
```

Parameters

target
The string to replace.

replacement
The string with which to replace *target*.

Return Value
A new string in which all occurrences of *target* in the receiver are replaced by *replacement*.

Discussion
Invokes `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1034)with `0` options and range of the whole string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1034)
- `stringByReplacingCharactersInRange:withString:` (page 1033)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1035)

Declared In
`NSString.h`

stringByReplacingOccurrencesOfString:withString:options:range:

Returns a new string in which all occurrences of a target string in a specified range of the receiver are replaced by another given string.

```
- (NSString *)stringByReplacingOccurrencesOfString:(NSString *)target
    withString:(NSString *)replacement options:(NSStringCompareOptions)options
    range:(NSRange)searchRange
```

Parameters
target
The string to replace.

replacement
The string with which to replace *target*.

options
A mask of options to use when comparing *target* with the receiver. Pass `0` to specify no options.

searchRange
The range in the receiver in which to search for *target*.

Return Value
A new string in which all occurrences of *target*, matched using *options*, in *searchRange* of the receiver are replaced by *replacement*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:` (page 1033)
- `stringByReplacingCharactersInRange:withString:` (page 1033)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1035)

Declared In
NSString.h

stringByReplacingPercentEscapesUsingEncoding:

Returns a new string made by replacing in the receiver all percent escapes with the matching characters as determined by a given encoding.

```
- (NSString *)stringByReplacingPercentEscapesUsingEncoding:(NSStringEncoding)encoding
```

Parameters

encoding

The encoding to use for the returned string.

Return Value
A new string made by replacing in the receiver all percent escapes with the matching characters as determined by the given encoding *encoding*. Returns `nil` if the transformation is not possible, for example, the percent escapes give a byte sequence not legal in *encoding*.

Discussion
See `CFURLCreateStringByReplacingPercentEscapes` for more complex transformations.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- stringByAddingPercentEscapesUsingEncoding:` (page 1026)

Declared In
NSURL.h

stringByResolvingSymlinksInPath

Returns a new string made from the receiver by resolving all symbolic links and standardizing path.

```
- (NSString *)stringByResolvingSymlinksInPath
```

Return Value
A new string made by expanding an initial tilde expression in the receiver, then resolving all symbolic links and references to current or parent directories if possible, to generate a standardized path. If the original path is absolute, all symbolic links are guaranteed to be removed; if it's a relative path, symbolic links that can't be resolved are left unresolved in the returned string. Returns `self` if an error occurs.

Discussion
If the name of the receiving path begins with `/private`, the `stringByResolvingSymlinksInPath` method strips off the `/private` designator, provided the result is the name of an existing file.

Note that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByStandardizingPath` (page 1036)
- `stringByExpandingTildeInPath` (page 1031)

Declared In
`NSPathUtilities.h`

stringByStandardizingPath

Returns a new string made by removing extraneous path components from the receiver.

```
- (NSString *)stringByStandardizingPath
```

Return Value
A new string made by removing extraneous path components from the receiver.

Discussion
If `stringByStandardizingPath` detects symbolic links in a pathname, the `stringByResolvingSymlinksInPath` (page 1035) method is called to resolve them. If an invalid pathname is provided, `stringByStandardizingPath` may attempt to resolve it by calling `stringByResolvingSymlinksInPath`, and the results are undefined. If any other kind of error is encountered (such as a path component not existing), `self` is returned.

This method can make the following changes in the provided string:

- Expand an initial tilde expression using `stringByExpandingTildeInPath` (page 1031).
- Reduce empty components and references to the current directory (that is, the sequences "`//`" and "`/./`") to single path separators.
- In absolute paths only, resolve references to the parent directory (that is, the component "..") to the real parent directory if possible using `stringByResolvingSymlinksInPath` (page 1035), which consults the file system to resolve each potential symbolic link.

  In relative paths, because symbolic links can't be resolved, references to the parent directory are left in place.
- Remove an initial component of "`/private`" from the path if the result still indicates an existing file or directory (checked by consulting the file system).

Note that the path returned by this method may still have symbolic link components in it. Note also that this method only works with file paths (not, for example, string representations of URLs).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1031)
- `stringByResolvingSymlinksInPath` (page 1035)

Declared In
`NSPathUtilities.h`

stringByTrimmingCharactersInSet:

Returns a new string made by removing from both ends of the receiver characters contained in a given character set.

```
- (NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set
```

Parameters

set

A character set containing the characters to remove from the receiver. *set* must not be `nil`.

Return Value

A new string made by removing from both ends of the receiver characters contained in *set*. If the receiver is composed entirely of characters from *set*, the empty string is returned.

Discussion

Use `whitespaceCharacterSet` (page 142) or `whitespaceAndNewlineCharacterSet` (page 142) to remove whitespace around strings.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- componentsSeparatedByCharactersInSet:` (page 985)

Declared In

`NSString.h`

stringsByAppendingPaths:

Returns an array of strings made by separately appending to the receiver each string in in a given array.

```
- (NSArray *)stringsByAppendingPaths:(NSArray *)paths
```

Parameters

paths

An array of `NSString` objects specifying paths to add to the receiver.

Return Value

An array of `NSString` objects made by separately appending each string in *paths* to the receiver, preceded if necessary by a path separator.

Discussion

Note that this method only works with file paths (not, for example, string representations of URLs). See `stringByAppendingPathComponent:` (page 1027) for an individual example.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPathUtilities.h`

substringFromIndex:

Returns a new string containing the characters of the receiver from the one at a given index to the end.

```
- (NSString *)substringFromIndex:(NSUInteger)anIndex
```

Parameters

anIndex

An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an `NSRangeException` if *anIndex* lies beyond the end of the receiver.

Return Value

A new string containing the characters of the receiver from the one at *anIndex* to the end. If *anIndex* is equal to the length of the string, returns an empty string.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `substringWithRange:` (page 1039)
- `substringToIndex:` (page 1038)

Declared In

`NSString.h`

substringToIndex:

Returns a new string containing the characters of the receiver up to, but not including, the one at a given index.

```
- (NSString *)substringToIndex:(NSUInteger)anIndex
```

Parameters

anIndex

An index. The value must lie within the bounds of the receiver, or be equal to the length of the receiver.

Important: Raises an `NSRangeException` if (*anIndex* - 1) lies beyond the end of the receiver.

Return Value

A new string containing the characters of the receiver up to, but not including, the one at *anIndex*. If *anIndex* is equal to the length of the string, returns a copy of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `substringFromIndex:` (page 1038)
- `substringWithRange:` (page 1039)

Declared In
NSString.h

substringWithRange:

Returns a string object containing the characters of the receiver that lie within a given range.

```
- (NSString *)substringWithRange:(NSRange)aRange
```

Parameters

aRange

A range. The range must not exceed the bounds of the receiver.

Important: Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver.

Return Value
A string object containing the characters of the receiver that lie within *aRange*.

Discussion
This method treats the length of the string as a valid range value that returns an empty string.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `substringFromIndex:` (page 1038)
- `substringToIndex:` (page 1038)

Declared In
NSString.h

uppercaseString

Returns an uppercased representation of the receiver.

```
- (NSString *)uppercaseString
```

Return Value
A string with each character from the receiver changed to its corresponding uppercase value.

Discussion
Case transformations aren't guaranteed to be symmetrical or to produce strings of the same lengths as the originals. See `lowercaseString` (page 1014) for an example.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `capitalizedString` (page 978)
- `lowercaseString` (page 1014)

Declared In
NSString.h

UTF8String

Returns a null-terminated UTF8 representation of the receiver.

```
- (const char *)UTF8String
```

Return Value
A null-terminated UTF8 representation of the receiver.

Discussion
The returned C string is automatically freed just as a returned object would be released; you should copy the C string if it needs to store it outside of the autorelease context in which the C string is created.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

writeToFile:atomically:encoding:error:

Writes the contents of the receiver to a file at a given path using a given encoding.

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

path
The file to which to write the receiver. If *path* contains a tilde (~) character, you must expand it with stringByExpandingTildeInPath (page 1031) before invoking this method.

useAuxiliaryFile
If YES, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *path*. If NO, the receiver is written directly to *path*. The YES option guarantees that *path*, if it exists at all, won't be corrupted even if the system should crash during writing.

enc
The encoding to use for the output.

error
If there is an error, upon return contains an NSError object that describes the problem. If you are not interested in details of errors, you may pass in NULL.

Return Value
YES if the file is written successfully, otherwise NO (if there was a problem writing to the file or with the encoding).

Discussion
This method overwrites any existing file at *path*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

writeToURL:atomically:encoding:error:

Writes the contents of the receiver to the URL specified by *url* using the specified encoding.

```
- (BOOL)writeToURL:(NSURL *)url atomically:(BOOL)useAuxiliaryFile
    encoding:(NSStringEncoding)enc error:(NSError **)error
```

Parameters

url

The URL to which to write the receiver.

useAuxiliaryFile

If YES, the receiver is written to an auxiliary file, and then the auxiliary file is renamed to *url*. If NO, the receiver is written directly to *url*. The YES option guarantees that *url*, if it exists at all, won't be corrupted even if the system should crash during writing.

The *useAuxiliaryFile* parameter is ignored if *url* is not of a type that can be accessed atomically.

enc

The encoding to use for the output.

error

If there is an error, upon return contains an NSError object that describes the problem. If you are not interested in details of errors, you may pass in NULL.

Return Value
YES if the URL is written successfully, otherwise NO (if there was a problem writing to the URL or with the encoding).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

Constants

unichar

Type for Unicode characters.

```
typedef unsigned short unichar;
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSString.h

NSStringCompareOptions

Type for string comparison options.

```
typedef NSUInteger NSStringCompareOptions;
```

Discussion

See "Search and Comparison Options" (page 1042) for possible values.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

Search and Comparison Options

These values represent the options available to many of the string classes' search and comparison methods.

```
enum {
    NSCaseInsensitiveSearch = 1,
    NSLiteralSearch = 2,
    NSBackwardsSearch = 4,
    NSAnchoredSearch = 8,
    NSNumericSearch = 64,
    NSDiacriticInsensitiveSearch = 128,
    NSWidthInsensitiveSearch = 256,
    NSForcedOrderingSearch = 512
};
```

Constants

`NSCaseInsensitiveSearch`

A case-insensitive search.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSLiteralSearch`

Exact character-by-character equivalence.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSBackwardsSearch`

Search from end of source string.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSAnchoredSearch`

Search is limited to start (or end, if `NSBackwardsSearch`) of source string.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSNumericSearch`
Numbers within strings are compared using numeric value, that is, `Foo2.txt` < `Foo7.txt` < `Foo25.txt`.

This option only applies to compare methods, not find.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSDiacriticInsensitiveSearch`
Search ignores diacritic marks.

For example, ‘ö’ is equal to ‘o’.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWidthInsensitiveSearch`
Search is ignores width differences ().

For example, ‘a’ is equal to UFF41.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSForcedOrderingSearch`
Comparisons are forced to return either `NSOrderedAscending` or `NSOrderedDescending` if the strings are equivalent but not strictly equal.

This option gives stability when sorting. For example, “aaa” is greater than "AAA” if `NSCaseInsensitiveSearch` is specified.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

Discussion
See Searching and Comparing Strings for details on the effects of these options.

Declared In
`NSString.h`

NSStringEncodingConversionOptions

Type for encoding conversion options.

```
typedef NSUInteger NSStringEncodingConversionOptions;
```

Discussion
See longLongValue (page 1014) for possible values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

Encoding Conversion Options

Options for converting string encodings.

```
enum {
    NSStringEncodingConversionAllowLossy = 1,
    NSStringEncodingConversionExternalRepresentation = 2
};
```

Constants

`NSStringEncodingConversionAllowLossy`

Allows lossy conversion.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSStringEncodingConversionExternalRepresentation`

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

Special Considerations

These constants are available in Mac OS X v10.4; they are, however, differently named:

```
typedef enum {
    NSAllowLossyEncodingConversion = 1,
    NSExternalRepresentationEncodingConversion = 2
} NSStringEncodingConversionOptions;
```

You can use them on Mac OS X v10.4 if you define the symbols as `extern` constants.

Declared In

`NSString.h`

NSString Handling Exception Names

These constants define the names of exceptions raised if `NSString` cannot represent a string in a given encoding, or parse a string as a property list.

```
extern NSString *NSParseErrorException;
extern NSString *NSCharacterConversionException;
```

Constants

`NSCharacterConversionException`

`NSString` raises an `NSCharacterConversionException` if a string cannot be represented in a file-system or string encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSParseErrorException`

`NSString` raises an `NSParseErrorException` if a string cannot be parsed as a property list.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

Declared In

`NSString.h`

NSStringEncoding

Type for string encoding.

```
typedef NSUInteger NSStringEncoding;
```

Discussion

See "String Encodings" (page 1045) for possible values.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSString.h`

String Encodings

The following constants are provided by `NSString` as possible string encodings.

```
enum {
    NSASCIIStringEncoding = 1,
    NSNEXTSTEPStringEncoding = 2,
    NSJapaneseEUCStringEncoding = 3,
    NSUTF8StringEncoding = 4,
    NSISOLatin1StringEncoding = 5,
    NSSymbolStringEncoding = 6,
    NSNonLossyASCIIStringEncoding = 7,
    NSShiftJISStringEncoding = 8,
    NSISOLatin2StringEncoding = 9,
    NSUnicodeStringEncoding = 10,
    NSWindowsCP1251StringEncoding = 11,
    NSWindowsCP1252StringEncoding = 12,
    NSWindowsCP1253StringEncoding = 13,
    NSWindowsCP1254StringEncoding = 14,
    NSWindowsCP1250StringEncoding = 15,
    NSISO2022JPStringEncoding = 21,
    NSMacOSRomanStringEncoding = 30,
    NSUTF16BigEndianStringEncoding = 0x90000100,
    NSUTF16LittleEndianStringEncoding = 0x94000100,
    NSUTF32StringEncoding = 0x8c000100,
    NSUTF32BigEndianStringEncoding = 0x98000100,
    NSUTF32LittleEndianStringEncoding = 0x9c000100,
};
```

Constants

`NSASCIIStringEncoding`

Strict 7-bit ASCII encoding within 8-bit chars; ASCII values 0…127 only.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSISO2022JPStringEncoding`

ISO 2022 Japanese encoding for email.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSISOLatin1StringEncoding`
8-bit ISO Latin 1 encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSISOLatin2StringEncoding`
8-bit ISO Latin 2 encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSJapaneseEUCStringEncoding`
8-bit EUC encoding for Japanese text.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSMacOSRomanStringEncoding`
Classic Macintosh Roman encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSNEXTSTEPStringEncoding`
8-bit ASCII encoding with NEXTSTEP extensions.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSNonLossyASCIIStringEncoding`
7-bit verbose ASCII to represent all Unicode characters.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSShiftJISStringEncoding`
8-bit Shift-JIS encoding for Japanese text.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSSymbolStringEncoding`
8-bit Adobe Symbol encoding vector.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF8StringEncoding`
An 8-bit representation of Unicode characters, suitable for transmission or storage by ASCII-based systems.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUnicodeStringEncoding`
The canonical Unicode encoding for string objects.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWindowsCP1250StringEncoding`
Microsoft Windows codepage 1250; equivalent to WinLatin2.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWindowsCP1251StringEncoding`
Microsoft Windows codepage 1251, encoding Cyrillic characters; equivalent to AdobeStandardCyrillic font encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWindowsCP1252StringEncoding`
Microsoft Windows codepage 1252; equivalent to WinLatin1.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWindowsCP1253StringEncoding`
Microsoft Windows codepage 1253, encoding Greek characters.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSWindowsCP1254StringEncoding`
Microsoft Windows codepage 1254, encoding Turkish characters.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF16BigEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF16LittleEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF32StringEncoding`
32-bit UTF encoding.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF32BigEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

`NSUTF32LittleEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in iPhone OS 2.0 and later.

Declared in `NSString.h`

Discussion
These values represent the various character encodings supported by the `NSString` classes. This is an incomplete list. Additional encodings are defined in *Strings Programming Guide for Core Foundation* (see `CFStringEncodingExt.h`); these encodings can be used with `NSString` by first passing the Core Foundation encoding to the `CFStringConvertEncodingToNSStringEncoding` function.

Declared In
`NSString.h`

NSThread Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSThread.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSThread` object controls a thread of execution. Use this class when you want to have an Objective-C method run in its own thread of execution. Threads are especially useful when you need to perform a lengthy task, but don't want it to block the execution of the rest of the application. In particular, you can use threads to avoid blocking the main thread of the application, which handles user interface and event-related actions. Threads can also be used to divide a large job into several smaller jobs, which can lead to performance increases on multi-core computers.

Prior to Mac OS X v10.5, the only way to start a new thread is to use the `detachNewThreadSelector:toTarget:withObject:` (page 1052) method. In Mac OS X v10.5 and later, you can create instances of `NSThread` and start them at a later time using the `start` (page 1062) method.

In Mac OS Xv10.5, the `NSThread` class supports semantics similar to those of `NSOperation` for monitoring the runtime condition of a thread. You can use these semantics to cancel the execution of a thread or determine if the thread is still executing or has finished its task. Canceling a thread requires support from your thread code; see the description for `cancel` (page 1057) for more information.

Subclassing Notes

In Mac OS X v10.5 and later, you can subclass `NSThread` and override the `main` method to implement your thread's main entry point. If you override `main`, you do not need to invoke the inherited behavior by calling `super`.

Tasks

Initializing an NSThread Object

- `init` (page 1057)
  Returns an initialized `NSThread` object.
- `initWithTarget:selector:object:` (page 1057)
  Returns an `NSThread` object initialized with the given arguments.

Starting a Thread

+ `detachNewThreadSelector:toTarget:withObject:` (page 1052)
  Detaches a new thread and uses the specified selector as the thread entry point.
- `start` (page 1062)
  Starts the receiver.
- `main` (page 1060)
  The main entry point routine for the thread.

Stopping a Thread

+ `sleepUntilDate:` (page 1056)
  Blocks the current thread until the time specified.
+ `sleepForTimeInterval:` (page 1055)
  Sleeps the process for a given time interval.
+ `exit` (page 1053)
  Terminates the current thread.
- `cancel` (page 1057)
  Changes the cancelled state of the receiver to indicate that it should exit.

Determining the Thread's Execution State

- `isExecuting` (page 1059)
  Returns a Boolean value that indicates whether the receiver is executing.
- `isFinished` (page 1059)
  Returns a Boolean value that indicates whether the receiver has finished execution.
- `isCancelled` (page 1058)
  Returns a Boolean value that indicates whether the receiver is cancelled.

Working with the Main Thread

+ `isMainThread` (page 1054)
  Returns a Boolean value that indicates whether the current thread is the main thread.
- `isMainThread` (page 1059)
  Returns a Boolean value that indicates whether the receiver is the main thread.
+ `mainThread` (page 1054)
  Returns the `NSThread` object representing the main thread.

Querying the Environment

+ `isMultiThreaded` (page 1054)
  Returns whether the application is multithreaded.
+ `currentThread` (page 1052)
  Returns the thread object representing the current thread of execution.
+ `callStackReturnAddresses` (page 1052)
  Returns an array containing the call stack return addresses.

Working with Thread Properties

- `threadDictionary` (page 1062)
  Returns the thread object's dictionary.
- `name` (page 1060)
  Returns the name of the receiver.
- `setName:` (page 1060)
  Sets the name of the receiver.
- `stackSize` (page 1061)
  Returns the stack size of the receiver.
- `setStackSize:` (page 1061)
  Sets the stack size of the receiver.

Working with Thread Priorities

`+ threadPriority` (page 1056)
Returns the current thread's priority.

`+ setThreadPriority:` (page 1055)
Sets the current thread's priority.

Class Methods

callStackReturnAddresses

Returns an array containing the call stack return addresses.

```
+ (NSArray *)callStackReturnAddresses
```

Return Value
An array containing the call stack return addresses. This value is `nil` by default.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSThread.h`

currentThread

Returns the thread object representing the current thread of execution.

```
+ (NSThread *)currentThread
```

Return Value
A thread object representing the current thread of execution.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ detachNewThreadSelector:toTarget:withObject:` (page 1052)

Declared In
`NSThread.h`

detachNewThreadSelector:toTarget:withObject:

Detaches a new thread and uses the specified selector as the thread entry point.

```
+ (void)detachNewThreadSelector:(SEL)aSelector toTarget:(id)aTarget
    withObject:(id)anArgument
```

Parameters

aSelector

The selector for the message to send to the target. This selector must take only one argument and must not have a return value.

aTarget

The object that will receive the message *aSelector* on the new thread.

anArgument

The single argument passed to the target. May be `nil`.

Discussion

For non garbage-collected applications, the method *aSelector* is responsible for setting up an autorelease pool for the newly detached thread and freeing that pool before it exits. Garbage-collected applications do not need to create an autorelease pool.

The objects *aTarget* and *anArgument* are retained during the execution of the detached thread, then released. The detached thread is exited (using the `exit` (page 1053) class method) as soon as *aTarget* has completed executing the *aSelector* method.

If this thread is the first thread detached in the application, this method posts the `NSWillBecomeMultiThreadedNotification` (page 1063) with object `nil` to the default notification center.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ currentThread` (page 1052)
`+ isMultiThreaded` (page 1054)
`- start` (page 1062)

Declared In

`NSThread.h`

exit

Terminates the current thread.

```
+ (void)exit
```

Discussion

This method uses the `currentThread` (page 1052) class method to access the current thread. Before exiting the thread, this method posts the `NSThreadWillExitNotification` (page 1063) with the thread being exited to the default notification center. Because notifications are delivered synchronously, all observers of `NSThreadWillExitNotification` (page 1063) are guaranteed to receive the notification before the thread exits.

Invoking this method should be avoided as it does not give your thread a chance to clean up any resources it allocated during its execution.

Availability

Available in iPhone OS 2.0 and later.

See Also
+ currentThread (page 1052)
+ sleepUntilDate: (page 1056)

Declared In
NSThread.h

isMainThread

Returns a Boolean value that indicates whether the current thread is the main thread.

```
+ (BOOL)isMainThread
```

Return Value
YES if the current thread is the main thread, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ mainThread (page 1054)

Declared In
NSThread.h

isMultiThreaded

Returns whether the application is multithreaded.

```
+ (BOOL)isMultiThreaded
```

Return Value
YES if the application is multithreaded, NO otherwise.

Discussion
An application is considered multithreaded if a thread was ever detached from the main thread using either detachNewThreadSelector:toTarget:withObject: (page 1052) or start (page 1062). If you detached a thread in your application using a non-Cocoa API, such as the POSIX or Multiprocessing Services APIs, this method could still return NO. The detached thread does not have to be currently running for the application to be considered multithreaded—this method only indicates whether a single thread has been spawned.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

mainThread

Returns the NSThread object representing the main thread.

```
+ (NSThread *)mainThread
```

Return Value
The `NSThread` object representing the main thread.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- isMainThread` (page 1059)

Declared In
`NSThread.h`

setThreadPriority:

Sets the current thread's priority.

```
+ (BOOL)setThreadPriority:(double)priority
```

Parameters

priority
The new priority, specified with a floating point number from 0.0 to 1.0, where 1.0 is highest priority.

Return Value
`YES` if the priority assignment succeeded, `NO` otherwise.

Discussion
The priorities in this range are mapped to the operating system's priority values.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ threadPriority` (page 1056)

Declared In
`NSThread.h`

sleepForTimeInterval:

Sleeps the process for a given time interval.

```
+ (void)sleepForTimeInterval:(NSTimeInterval)ti
```

Parameters

ti
The duration of the sleep.

Discussion
No run loop processing occurs while the thread is blocked.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

sleepUntilDate:

Blocks the current thread until the time specified.

```
+ (void)sleepUntilDate:(NSDate *)aDate
```

Parameters

aDate

The time at which to resume processing.

Discussion
No run loop processing occurs while the thread is blocked.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ currentThread (page 1052)
+ exit (page 1053)

Declared In
NSThread.h

threadPriority

Returns the current thread's priority.

```
+ (double)threadPriority
```

Return Value
The current thread's priority, which is specified by a floating point number from 0.0 to 1.0, where 1.0 is highest priority.

Discussion
The priorities in this range are mapped to the operating system's priority values. A "typical" thread priority might be 0.5, but because the priority is determined by the kernel, there is no guarantee what this value actually will be.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ setThreadPriority: (page 1055)

Declared In
NSThread.h

Instance Methods

cancel

Changes the cancelled state of the receiver to indicate that it should exit.

```
- (void)cancel
```

Discussion

The semantics of this method are the same as those used for the `NSOperation` object. This method sets state information in the receiver that is then reflected by the `isCancelled` method. Threads that support cancellation should periodically call the `isCancelled` method to determine if the thread has in fact been cancelled, and exit if it has been.

For more information about cancellation and operation objects, see *NSOperation Class Reference*.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `isCancelled` (page 1058)

Declared In

`NSThread.h`

init

Returns an initialized `NSThread` object.

```
- (id)init
```

Return Value

An initialized `NSThread` object.

Discussion

This is the designated initializer for `NSThread`.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `initWithTarget:selector:object:` (page 1057)
- `start` (page 1062)

Declared In

`NSThread.h`

initWithTarget:selector:object:

Returns an `NSThread` object initialized with the given arguments.

```
- (id)initWithTarget:(id)target
    selector:(SEL)selector
    object:(id)argument
```

Parameters

target

The object to which the message specified by *selector* is sent.

selector

The selector for the message to send to *target*. This selector must take only one argument and must not have a return value.

argument

The single argument passed to the target. May be nil.

Return Value

An NSThread object initialized with the given arguments.

Discussion

For non garbage-collected applications, the method *selector* is responsible for setting up an autorelease pool for the newly detached thread and freeing that pool before it exits. Garbage-collected applications do not need to create an autorelease pool.

The objects *target* and *argument* are retained during the execution of the detached thread. They are released when the thread finally exits.

Availability

Available in iPhone OS 2.0 and later.

See Also

- init (page 1057)
- start (page 1062)

Declared In

NSThread.h

isCancelled

Returns a Boolean value that indicates whether the receiver is cancelled.

```
- (BOOL)isCancelled
```

Return Value

YES if the receiver has been cancelled, otherwise NO.

Discussion

If your thread supports cancellation, it should call this method periodically and exit if it ever returns YES.

Availability

Available in iPhone OS 2.0 and later.

See Also

- cancel (page 1057)
- isExecuting (page 1059)
- isFinished (page 1059)

Declared In
NSThread.h

isExecuting

Returns a Boolean value that indicates whether the receiver is executing.

```
- (BOOL)isExecuting
```

Return Value
YES if the receiver is executing, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isCancelled (page 1058)
- isFinished (page 1059)

Declared In
NSThread.h

isFinished

Returns a Boolean value that indicates whether the receiver has finished execution.

```
- (BOOL)isFinished
```

Return Value
YES if the receiver has finished execution, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isCancelled (page 1058)
- isExecuting (page 1059)

Declared In
NSThread.h

isMainThread

Returns a Boolean value that indicates whether the receiver is the main thread.

```
- (BOOL)isMainThread
```

Return Value
YES if the receiver is the main thread, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

main

The main entry point routine for the thread.

```
- (void)main
```

Discussion
The default implementation of this method takes the target and selector used to initialize the receiver and invokes the selector on the specified target. If you subclass `NSThread`, you can override this method and use it to implement the main body of your thread instead. If you do so, you do not need to invoke `super`.

You should never invoke this method directly. You should always start your thread by invoking the `start` method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- start` (page 1062)

Declared In
NSThread.h

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setName:` (page 1060)

Declared In
NSThread.h

setName:

Sets the name of the receiver.

```
- (void)setName:(NSString *)n
```

Parameters

n

The name for the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `name` (page 1060)

Declared In

`NSThread.h`

setStackSize:

Sets the stack size of the receiver.

```
- (void)setStackSize:(NSUInteger)s
```

Parameters

s

The stack size for the receiver. This value must be a multiple of 4KB.

Discussion

You must call this method before starting your thread. Setting the stack size after the thread has started changes the attribute size (which is reflected by the `stackSize` (page 1061) method), but it does not affect the actual number of pages set aside for the thread.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `stackSize` (page 1061)

Declared In

`NSThread.h`

stackSize

Returns the stack size of the receiver.

```
- (NSUInteger)stackSize
```

Return Value

The stack size of the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `setStackSize:` (page 1061)

Declared In

`NSThread.h`

start

Starts the receiver.

```
- (void)start
```

Discussion

This method spawns the new thread and invokes the receiver's `main` method on the new thread. If you initialized the receiver with a target and selector, the default `main` method invokes that selector automatically.

If this thread is the first thread detached in the application, this method posts the `NSWillBecomeMultiThreadedNotification` (page 1063) with object `nil` to the default notification center.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `init` (page 1057)
- `initWithTarget:selector:object:` (page 1057)
- `main` (page 1060)

Declared In

`NSThread.h`

threadDictionary

Returns the thread object's dictionary.

```
- (NSMutableDictionary *)threadDictionary
```

Return Value

The thread object's dictionary.

Discussion

You can use the returned dictionary to store thread-specific data. The thread dictionary is not used during any manipulations of the `NSThread` object—it is simply a place where you can store any interesting data. For example, Foundation uses it to store the thread's default `NSConnection` and `NSAssertionHandler` instances. You may define your own keys for the dictionary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSThread.h`

Notifications

NSDidBecomeSingleThreadedNotification

Not implemented.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

NSThreadWillExitNotification

An NSThread object posts this notification when it receives the exit (page 1053) message, before the thread exits. Observer methods invoked to receive this notification execute in the exiting thread, before it exits.

The notification object is the exiting NSThread object. This notification does not contain a *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

NSWillBecomeMultiThreadedNotification

Posted when the first thread is detached from the current thread. The NSThread class posts this notification at most once—the first time a thread is detached using detachNewThreadSelector:toTarget:withObject: (page 1052) or the start (page 1062) method. Subsequent invocations of those methods do not post this notification. Observers of this notification have their notification method invoked in the main thread, not the new thread. The observer notification methods always execute before the new thread begins executing.

This notification does not contain a notification object or a *userInfo* dictionary.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSThread.h

NSTimer Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSTimer.h
Companion guides:	Timers Run Loops

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSTimer` creates timer objects or, more simply, timers. A timer waits until a certain time interval has elapsed and then fires, sending a specified message to a specified object. For example, you could create an `NSTimer` object that sends a message to a window, telling it to update itself after a certain time interval.

Timers work in conjunction with run loops. To use a timer effectively, you should be aware of how run loops operate—see `NSRunLoop` and *Run Loops*. Note in particular that run loops retain their timers, so you can release a timer after you have added it to a run loop. Moreover, timers may not fire exactly when scheduled.

There are three ways to create a timer. The `scheduledTimerWithTimeInterval:invocation:repeats:` (page 1067) and `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1067) class methods automatically add the new timer to the current `NSRunLoop` object in the default mode (`NSDefaultRunLoopMode`). The `timerWithTimeInterval:invocation:repeats:` (page 1068) and `timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1068) class methods create timers that you may add to a run loop at a later time by sending the message `addTimer:forMode:` (page 891) to the `NSRunLoop` object. Finally, you can allocate the timer directly and initialize it with `initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1070), which allows you to specify both an initial fire date and a repeating interval. If you specify that the timer should repeat, it automatically reschedules itself after it fires. If you specify that the timer should not repeat, it is automatically invalidated after it fires.

To request the removal of a timer from an `NSRunLoop` object, send the timer the `invalidate` (page 1070) message from the same thread on which the timer was installed. This message immediately disables the timer, so it no longer affects the `NSRunLoop` object. The run loop removes and releases the timer, either just before the `invalidate` (page 1070) method returns or at some later point.

`NSTimer` is “toll-free bridged” with its Core Foundation counterpart, CFRunLoopTimer. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSTimer *` parameter, you can pass a `CFRunLoopTimerRef`, and in a function where you see a `CFRunLoopTimerRef` parameter, you can pass an `NSTimer` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Tasks

Creating a Timer

`+ scheduledTimerWithTimeInterval:invocation:repeats:` (page 1067)
Returns a new `NSTimer` object and adds it to the current `NSRunLoop` object in the default mode.

`+ scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1067)
Returns a new `NSTimer` object and adds it to the current `NSRunLoop` object in the default mode.

`+ timerWithTimeInterval:invocation:repeats:` (page 1068)
Returns a new `NSTimer` that, when added to a run loop, will fire after *`seconds`*.

`+ timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1068)
Returns a new `NSTimer` that, when added to a run loop, will fire after *`seconds`*.

`- initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1070)
Initializes a new `NSTimer` that, when added to a run loop, will fire at *`date`* and then, if *`repeats`* is `YES`, every *`seconds`* after that.

Firing a Timer

`- fire` (page 1069)
Causes the receiver’s message to be sent to its target.

Stopping a Timer

- `invalidate` (page 1070)
  Stops the receiver from ever firing again and requests its removal from its `NSRunLoop` object.

Information About a Timer

- `isValid` (page 1071)
  Returns a Boolean value that indicates whether the receiver is currently valid.
- `fireDate` (page 1069)
  Returns the date at which the receiver will fire.
- `setFireDate:` (page 1071)
  Resets the receiver to fire next at a given date.
- `timeInterval` (page 1071)
  Returns the receiver's time interval.
- `userInfo` (page 1071)
  Returns the receiver's *`userInfo`* object.

Class Methods

scheduledTimerWithTimeInterval:invocation:repeats:

Returns a new `NSTimer` object and adds it to the current `NSRunLoop` object in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    invocation:(NSInvocation *)invocation repeats:(BOOL)repeats
```

Discussion

After *`seconds`* have elapsed, the timer fires, sending the message in *`invocation`* to its target. If *`seconds`* is less than or equal to 0.0, this method chooses a nonnegative interval. If *`repeats`* is `YES`, the timer will repeatedly reschedule itself until invalidated. If *`repeats`* is `NO`, the timer will be invalidated after it fires.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSTimer.h`

scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:

Returns a new `NSTimer` object and adds it to the current `NSRunLoop` object in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds target:(id)target
    selector:(SEL)aSelector userInfo:(id)userInfo repeats:(BOOL)repeats
```

Discussion
After *seconds* have elapsed, the timer fires, sending the message *aSelector* to *target*. The *aSelector* method has the following syntax:

```
- (void)myTimerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to *aSelector*. To pass more information to the target, use *userInfo*. The target gets *userInfo* by sending `userInfo` (page 1071) to the timer.

If *seconds* is less than or equal to 0.0, this method chooses a nonnegative interval. If *repeats* is `YES`, the timer will repeatedly reschedule itself until invalidated. If *repeats* is `NO`, the timer will be invalidated after it fires.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSTimer.h`

timerWithTimeInterval:invocation:repeats:

Returns a new `NSTimer` that, when added to a run loop, will fire after *seconds*.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds invocation:(NSInvocation
     *)invocation repeats:(BOOL)repeats
```

Discussion
If *seconds* is less than or equal to 0.0, this method chooses a nonnegative interval. Upon firing, the timer sends the message in *invocation* to its target. If *repeats* is `YES`, the timer will repeatedly reschedule itself until invalidated. If *repeats* is `NO`, the timer will be invalidated after it fires.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSTimer.h`

timerWithTimeInterval:target:selector:userInfo:repeats:

Returns a new `NSTimer` that, when added to a run loop, will fire after *seconds*.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds target:(id)target
    selector:(SEL)aSelector userInfo:(id)userInfo repeats:(BOOL)repeats
```

Discussion
Upon firing, the timer sends *aSelector* to *target*. The *aSelector* method has the following syntax:

```
- (void)myTimerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to *aSelector*. To pass more information to the target, use *userInfo*. The target gets *userInfo* by sending `userInfo` (page 1071) to the timer.

If *seconds* is less than or equal to 0.0, this method chooses a nonnegative interval. If *repeats* is YES, the timer will repeatedly reschedule itself until invalidated. If *repeats* is NO, the timer will be invalidated after it fires.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimer.h

Instance Methods

fire

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

```
- (void)fire
```

Discussion
If the timer is non-repeating, it is automatically invalidated after firing, even if its scheduled fire date has not arrived. A repeating timer can be fired with this method without interrupting its regular firing schedule.

Availability
Available in iPhone OS 2.0 and later.

See Also
- invalidate (page 1070)

Declared In
NSTimer.h

fireDate

Returns the date at which the receiver will fire.

```
- (NSDate *)fireDate
```

Return Value
The date at which the receiver will fire. If the timer is no longer valid, this method returns the last date at which the timer fired.

Discussion
Use isValid (page 1071) to verify that the timer is valid.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setFireDate: (page 1071)

Declared In
NSTimer.h

initWithFireDate:interval:target:selector:userInfo:repeats:

Initializes a new NSTimer that, when added to a run loop, will fire at *date* and then, if *repeats* is YES, every *seconds* after that.

```
- (id)initWithFireDate:(NSDate *)date interval:(NSTimeInterval)seconds
    target:(id)target selector:(SEL)aSelector userInfo:(id)userInfo
    repeats:(BOOL)repeats
```

Discussion
Upon firing, the timer sends *aSelector* to *target*. The *aSelector* method has the following syntax:

```
- (void)myTimerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to *aSelector*. To pass more information to the target, use *userInfo*. The target gets *userInfo* by sending userInfo (page 1071) to the timer.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimer.h

invalidate

Stops the receiver from ever firing again and requests its removal from its NSRunLoop object.

```
- (void)invalidate
```

Discussion
This is the only way to remove a timer from an NSRunLoop object. The NSRunLoop object removes and releases the timer, either just before the invalidate (page 1070) method returns or at some later point.

You must send this message from the thread on which the timer was installed. If you send this message from another thread, the input source associated with the timer may not be removed from its run loop, which could prevent the thread from exiting properly.

The receiver releases its references to the *target* and *userInfo* objects at the point of invalidation.

Availability
Available in iPhone OS 2.0 and later.

See Also
- fire (page 1069)

Declared In
NSTimer.h

isValid

Returns a Boolean value that indicates whether the receiver is currently valid.

```
- (BOOL)isValid
```

Return Value

`YES` if the receiver is currently valid, otherwise `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSTimer.h`

setFireDate:

Resets the receiver to fire next at a given date.

```
- (void)setFireDate:(NSDate *)date
```

Parameters

date

The date at which to fire the receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- fireDate` (page 1069)

Declared In

`NSTimer.h`

timeInterval

Returns the receiver's time interval.

```
- (NSTimeInterval)timeInterval
```

Return Value

The receiver's time interval. If the receiver is a non-repeating timer, returns `0` (even if a time interval was set).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSTimer.h`

userInfo

Returns the receiver's *userInfo* object.

```
- (id)userInfo
```

Return Value
The receiver's *userInfo* object, containing additional data the target may use when the receiver is fired.

Discussion
Do not invoke this method after the timer is invalidated. Use `isValid` (page 1071) to test whether the timer is valid.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1067)
`+ timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1068)
`- invalidate` (page 1070)

Declared In
`NSTimer.h`

NSTimeZone Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSTimeZone.h
Companion guide:	Date and Time Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

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

`NSTimeZone` provides several class methods to get time zone objects: `timeZoneWithName:` (page 1080), `timeZoneWithName:data:` (page 1081), `timeZoneWithAbbreviation:` (page 1080), and `timeZoneForSecondsFromGMT:` (page 1079). The class also permits you to set the default time zone *within your application* (`setDefaultTimeZone:` (page 1078)). You can access this default time zone at

any time with the `defaultTimeZone` (page 1077) class method, and with the `localTimeZone` (page 1078) class method, you can get a relative time zone object that decodes itself to become the default time zone for any locale in which it finds itself.

Cocoa does not provide any API to change the time zone of the computer, or of other applications.

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

Note that, strictly, time zone database entries such as "America/Los_Angeles" are IDs not names. An example of a time zone name is "Pacific Daylight Time". Although many `NSTimeZone` method names include the word "name", they refer to IDs.

`NSTimeZone` is "toll-free bridged" with its Core Foundation counterpart, *CFTimeZone Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSTimeZone *` parameter, you can pass a `CFTimeZoneRef`, and in a function where you see a `CFTimeZoneRef` parameter, you can pass an `NSTimeZone` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1246)
- `- initWithCoder:` (page 1246)

NSCopying
- `- copyWithZone:` (page 1250)

Tasks

Creating and Initializing Time Zone Objects

`+ timeZoneWithAbbreviation:` (page 1080)
Returns the time zone object identified by a given abbreviation.

`+ timeZoneWithName:` (page 1080)
Returns the time zone object identified by a given ID.

`+ timeZoneWithName:data:` (page 1081)
Returns the time zone with a given ID whose data has been initialized using given data,

`+ timeZoneForSecondsFromGMT:` (page 1079)
Returns a time zone object offset from Greenwich Mean Time by a given number of seconds.

`- initWithName:` (page 1084)
Returns a time zone initialized with a given ID.

- initWithName:data: (page 1084)
  Initializes a time zone with a given ID and time zone data.

Working with System Time Zones

+ localTimeZone (page 1078)
  Returns an object that forwards all messages to the default time zone for the current application.

+ defaultTimeZone (page 1077)
  Returns the default time zone for the current application.

+ setDefaultTimeZone: (page 1078)
  Sets the default time zone for the current application to a given time zone.

+ resetSystemTimeZone (page 1078)
  Resets the system time zone object cached by the application, if any.

+ systemTimeZone (page 1079)
  Returns the time zone currently used by the system.

Getting Time Zone Information

+ abbreviationDictionary (page 1076)
  Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

+ knownTimeZoneNames (page 1077)
  Returns an array of strings listing the IDs of all the time zones known to the system.

Getting Information About a Specific Time Zone

- abbreviation (page 1082)
  Returns the abbreviation for the receiver.

- abbreviationForDate: (page 1082)
  Returns the abbreviation for the receiver at a given date.

- name (page 1086)
  Returns the geopolitical region ID that identifies the receiver.

- secondsFromGMT (page 1088)
  Returns the current difference in seconds between the receiver and Greenwich Mean Time.

- secondsFromGMTForDate: (page 1088)
  Returns the difference in seconds between the receiver and Greenwich Mean Time at a given date.

- data (page 1082)
  Returns the data that stores the information used by the receiver.

Comparing Time Zones

- `isEqualToTimeZone:` (page 1086)
  Returns a Boolean value that indicates whether the receiver has the same name and data as another given time zone.

Describing a Time Zone

- `description` (page 1084)
  Returns the description of the receiver.
- `localizedName:locale:` (page 1086)
  Returns the name of the receiver localized for a given locale.

Getting Information About Daylight Saving

- `isDaylightSavingTime` (page 1085)
  Returns a Boolean value that indicates whether the receiver is currently using daylight saving time.
- `daylightSavingTimeOffset` (page 1083)
  Returns the current daylight saving time offset of the receiver.
- `isDaylightSavingTimeForDate:` (page 1085)
  Returns a Boolean value that indicates whether the receiver uses daylight savings time at a given date.
- `daylightSavingTimeOffsetForDate:` (page 1083)
  Returns the daylight saving time offset for a given date.
- `nextDaylightSavingTimeTransition` (page 1087)
  Returns the date of the next daylight saving time transition for the receiver.
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1087)
  Returns the next daylight saving time transition after a given date.

Class Methods

abbreviationDictionary

Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

```
+ (NSDictionary *)abbreviationDictionary
```

Return Value

A dictionary holding the mappings of time zone abbreviations to time zone names.

Discussion

Note that more than one time zone may have the same abbreviation—for example, US/Pacific and Canada/Pacific both use the abbreviation “PST.” In these cases, `abbreviationDictionary` chooses a single name to map the abbreviation to.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

defaultTimeZone

Returns the default time zone for the current application.

```
+ (NSTimeZone *)defaultTimeZone
```

Return Value
The default time zone for the current application. If no default time zone has been set, this method invokes `systemTimeZone` (page 1079) and returns the system time zone.

Discussion
The default time zone is the one that the application is running with, which you can change (so you can make the application run as if it were in a different time zone).

If you get the default time zone and hold onto the returned object, it does not change if a subsequent invocation of `setDefaultTimeZone:` (page 1078) changes the default time zone—you still have the specific time zone you originally got. Contrast this behavior with the object returned by `localTimeZone` (page 1078).

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ localTimeZone` (page 1078)
`+ setDefaultTimeZone:` (page 1078)
`+ systemTimeZone` (page 1079)

Declared In
NSTimeZone.h

knownTimeZoneNames

Returns an array of strings listing the IDs of all the time zones known to the system.

```
+ (NSArray *)knownTimeZoneNames
```

Return Value
An array of strings listing the IDs of all the time zones known to the system.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

localTimeZone

Returns an object that forwards all messages to the default time zone for the current application.

```
+ (NSTimeZone *)localTimeZone
```

Return Value

An object that forwards all messages to the default time zone for the current application.

Discussion

The local time zone represents the current state of the default time zone at all times. If you get the *default* time zone (using `defaultTimeZone` (page 1077)) and hold onto the returned object, it does not change if a subsequent invocation of `setDefaultTimeZone:` (page 1078) changes the default time zone—you still have the specific time zone you originally got. The *local* time zone adds a level of indirection, it acts as if it were the current default time zone whenever you invoke a method on it.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ defaultTimeZone` (page 1077)

`+ setDefaultTimeZone:` (page 1078)

Declared In

`NSTimeZone.h`

resetSystemTimeZone

Resets the system time zone object cached by the application, if any.

```
+ (void)resetSystemTimeZone
```

Discussion

If the application has cached the system time zone, this method clears that cached object. If you subsequently invoke `systemTimeZone` (page 1079), `NSTimeZone` will attempt to redetermine the system time zone and a new object will be created and cached (see `systemTimeZone` (page 1079)).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ systemTimeZone` (page 1079)

Declared In

`NSTimeZone.h`

setDefaultTimeZone:

Sets the default time zone for the current application to a given time zone.

```
+ (void)setDefaultTimeZone:(NSTimeZone *)aTimeZone
```

Parameters

aTimeZone

The new default time zone for the current application.

Discussion

There can be only one default time zone, so by setting a new default time zone, you lose the previous one.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ defaultTimeZone (page 1077)

+ localTimeZone (page 1078)

Declared In

NSTimeZone.h

systemTimeZone

Returns the time zone currently used by the system.

```
+ (NSTimeZone *)systemTimeZone
```

Return Value

The time zone currently used by the system. If the current time zone cannot be determined, returns the GMT time zone.

Special Considerations

If you get the system time zone, it is cached by the application and does not change if the user subsequently changes the system time zone. The next time you invoke `systemTimeZone`, you get back the same time zone you originally got. You have to invoke `resetSystemTimeZone` (page 1078) to clear the cached object.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ resetSystemTimeZone (page 1078)

Declared In

NSTimeZone.h

timeZoneForSecondsFromGMT:

Returns a time zone object offset from Greenwich Mean Time by a given number of seconds.

```
+ (id)timeZoneForSecondsFromGMT:(NSInteger)seconds
```

Parameters

seconds

The number of seconds by which the new time zone is offset from GMT.

Return Value
A time zone object offset from Greenwich Mean Time by *seconds*.

Discussion
The name of the new time zone is GMT +/– the offset, in hours and minutes. Time zones created with this method never have daylight savings, and the offset is constant no matter the date.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ timeZoneWithAbbreviation: (page 1080)
+ timeZoneWithName: (page 1080)

Declared In
NSTimeZone.h

timeZoneWithAbbreviation:

Returns the time zone object identified by a given abbreviation.

```
+ (id)timeZoneWithAbbreviation:(NSString *)abbreviation
```

Parameters
abbreviation
An abbreviation for a time zone.

Return Value
The time zone object identified by *abbreviation* determined by resolving the abbreviation to a name using the abbreviation dictionary and then returning the time zone for that name. Returns nil if there is no match for *abbreviation*.

Discussion
In general, you are discouraged from using abbreviations except for unique instances such as “UTC” or “GMT”. Time Zone abbreviations are not standardized and so a given abbreviation may have multiple meanings—for example, “EST” refers to Eastern Time in both the United States and Australia

Availability
Available in iPhone OS 2.0 and later.

See Also
+ abbreviationDictionary (page 1076)
+ timeZoneForSecondsFromGMT: (page 1079)
+ timeZoneWithName: (page 1080)

Declared In
NSTimeZone.h

timeZoneWithName:

Returns the time zone object identified by a given ID.

```
+ (id)timeZoneWithName:(NSString *)aTimeZoneName
```

Parameters

aName

The ID for the time zone.

Return Value

The time zone in the information directory with a name matching *aName*. Returns `nil` if there is no match for the name.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ timeZoneForSecondsFromGMT:` (page 1079)

`+ timeZoneWithAbbreviation:` (page 1080)

`+ knownTimeZoneNames` (page 1077)

Declared In

`NSTimeZone.h`

timeZoneWithName:data:

Returns the time zone with a given ID whose data has been initialized using given data,

```
+ (id)timeZoneWithName:(NSString *)aTimeZoneName data:(NSData *)data
```

Parameters

aTimeZoneName

The ID for the time zone.

data

The data from the time-zone files located at `/usr/share/zoneinfo`.

Return Value

The time zone with the ID *aTimeZoneName* whose data has been initialized using the contents of *data*.

Discussion

You should not call this method directly—use `timeZoneWithName:` (page 1080) to get the time zone object for a given name.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ timeZoneWithName:` (page 1080)

Declared In

`NSTimeZone.h`

Instance Methods

abbreviation

Returns the abbreviation for the receiver.

```
- (NSString *)abbreviation
```

Return Value
The abbreviation for the receiver, such as “EDT” (Eastern Daylight Time).

Discussion
Invokes `abbreviationForDate:` (page 1082) with the current date as the argument.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSTimeZone.h`

abbreviationForDate:

Returns the abbreviation for the receiver at a given date.

```
- (NSString *)abbreviationForDate:(NSDate *)aDate
```

Parameters
aDate

The date for which to get the abbreviation for the receiver.

Return Value
The abbreviation for the receiver at *aDate*.

Discussion
Note that the abbreviation may be different at different dates. For example, during daylight savings time the US/Eastern time zone has an abbreviation of “EDT.” At other times, its abbreviation is “EST.”

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSTimeZone.h`

data

Returns the data that stores the information used by the receiver.

```
- (NSData *)data
```

Return Value
The data that stores the information used by the receiver.

Discussion
This data should be treated as an opaque object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

daylightSavingTimeOffset

Returns the current daylight saving time offset of the receiver.

```
- (NSTimeInterval)daylightSavingTimeOffset
```

Return Value
The daylight current saving time offset of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isDaylightSavingTime (page 1085)
- isDaylightSavingTimeForDate: (page 1085)
- daylightSavingTimeOffsetForDate: (page 1083)

Declared In
NSTimeZone.h

daylightSavingTimeOffsetForDate:

Returns the daylight saving time offset for a given date.

```
- (NSTimeInterval)daylightSavingTimeOffsetForDate:(NSDate *)aDate
```

Parameters
aDate
A date.

Return Value
The daylight saving time offset for *aDate*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isDaylightSavingTime (page 1085)
- daylightSavingTimeOffset (page 1083)
- isDaylightSavingTimeForDate: (page 1085)
- nextDaylightSavingTimeTransitionAfterDate: (page 1087)

Declared In
NSTimeZone.h

description

Returns the description of the receiver.

```
- (NSString *)description
```

Return Value

The description of the receiver, including the name, abbreviation, offset from GMT, and whether or not daylight savings time is currently in effect.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSTimeZone.h

initWithName:

Returns a time zone initialized with a given ID.

```
- (id)initWithName:(NSString *)aName
```

Parameters

aName

The ID for the time zone.

Return Value

A time zone object initialized with the ID *aName*.

Discussion

If *aName* is a known ID, this method calls `initWithName:data:` (page 1084) with the appropriate data object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSTimeZone.h

initWithName:data:

Initializes a time zone with a given ID and time zone data.

```
- (id)initWithName:(NSString *)aName data:(NSData *)data
```

Parameters

aName

The ID for the time zone.

data

The data from the time-zone files located at `/usr/share/zoneinfo`.

Discussion

You should not call this method directly—use `initWithName:` (page 1084) to get a time zone object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

isDaylightSavingTime

Returns a Boolean value that indicates whether the receiver is currently using daylight saving time.

```
- (BOOL)isDaylightSavingTime
```

Return Value
YES if the receiver is currently using daylight savings time, otherwise NO.

Discussion
This method invokes isDaylightSavingTimeForDate: (page 1085) with the current date as the argument.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isDaylightSavingTimeForDate: (page 1085)
- daylightSavingTimeOffset (page 1083)
- daylightSavingTimeOffsetForDate: (page 1083)
- nextDaylightSavingTimeTransition (page 1087)
- nextDaylightSavingTimeTransitionAfterDate: (page 1087)

Declared In
NSTimeZone.h

isDaylightSavingTimeForDate:

Returns a Boolean value that indicates whether the receiver uses daylight savings time at a given date.

```
- (BOOL)isDaylightSavingTimeForDate:(NSDate *)aDate
```

Parameters
aDate
The date against which to test the receiver.

Return Value
YES if the receiver uses daylight savings time at *aDate*, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- isDaylightSavingTime (page 1085)
- daylightSavingTimeOffset (page 1083)
- daylightSavingTimeOffsetForDate: (page 1083)

- nextDaylightSavingTimeTransitionAfterDate: (page 1087)

Declared In
NSTimeZone.h

isEqualToTimeZone:

Returns a Boolean value that indicates whether the receiver has the same name and data as another given time zone.

```
- (BOOL)isEqualToTimeZone:(NSTimeZone *)aTimeZone
```

Parameters

aTimeZone
The time zone to compare with the receiver.

Return Value
YES if *aTimeZone* and the receiver have the same name and data, otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

localizedName:locale:

Returns the name of the receiver localized for a given locale.

```
- (NSString *)localizedName:(NSTimeZoneNameStyle)style locale:(NSLocale *)locale
```

Parameters

style
The format style for the returned string.

locale
The locale for which to format the name.

Return Value
The name of the receiver localized for *locale* using *style*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

name

Returns the geopolitical region ID that identifies the receiver.

```
- (NSString *)name
```

Return Value
The geopolitical region ID that identifies the receiver.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

nextDaylightSavingTimeTransition

Returns the date of the next daylight saving time transition for the receiver.

```
- (NSDate *)nextDaylightSavingTimeTransition
```

Return Value
The date of the next (after the current instant) daylight saving time transition for the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- isDaylightSavingTime` (page 1085)
- `- isDaylightSavingTimeForDate:` (page 1085)
- `- nextDaylightSavingTimeTransitionAfterDate:` (page 1087)

Declared In
NSTimeZone.h

nextDaylightSavingTimeTransitionAfterDate:

Returns the next daylight saving time transition after a given date.

```
- (NSDate *)nextDaylightSavingTimeTransitionAfterDate:(NSDate *)aDate
```

Parameters
aDate
A date.

Return Value
The next daylight saving time transition after *aDate*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `- isDaylightSavingTime` (page 1085)
- `- isDaylightSavingTimeForDate:` (page 1085)
- `- nextDaylightSavingTimeTransition` (page 1087)

Declared In
NSTimeZone.h

secondsFromGMT

Returns the current difference in seconds between the receiver and Greenwich Mean Time.

```
- (NSInteger)secondsFromGMT
```

Return Value
The current difference in seconds between the receiver and Greenwich Mean Time.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

secondsFromGMTForDate:

Returns the difference in seconds between the receiver and Greenwich Mean Time at a given date.

```
- (NSInteger)secondsFromGMTForDate:(NSDate *)aDate
```

Parameters

aDate

The date against which to test the receiver.

Return Value
The difference in seconds between the receiver and Greenwich Mean Time at *aDate*.

Discussion
The difference may be different from the current difference if the time zone changes its offset from GMT at different points in the year—for example, the U.S. time zones change with daylight savings time.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

Constants

NSTimeZoneNameStyle

Defines a type for time zone name styles.

```
typedef NSInteger NSTimeZoneNameStyle;
```

Discussion
See "Time Zone Name Styles" (page 1089) for possible values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

Time Zone Name Styles

Specify styles for presenting time zone names.

```
enum {
    NSTimeZoneNameStyleStandard,
    NSTimeZoneNameStyleShortStandard,
    NSTimeZoneNameStyleDaylightSaving,
    NSTimeZoneNameStyleShortDaylightSaving
};
```

Constants

`NSTimeZoneNameStyleStandard`
Specifies a standard name style.
Available in iPhone OS 2.0 and later.
Declared in `NSTimeZone.h`

`NSTimeZoneNameStyleShortStandard`
Specifies a short name style.
Available in iPhone OS 2.0 and later.
Declared in `NSTimeZone.h`

`NSTimeZoneNameStyleDaylightSaving`
Specifies a daylight saving name style.
Available in iPhone OS 2.0 and later.
Declared in `NSTimeZone.h`

`NSTimeZoneNameStyleShortDaylightSaving`
Specifies a short daylight saving name style.
Available in iPhone OS 2.0 and later.
Declared in `NSTimeZone.h`

Declared In
NSTimeZone.h

Notifications

NSSystemTimeZoneDidChangeNotification

Sent when the time zone changed.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSTimeZone.h

NSURL Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSURLHandleClient NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURL.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The NSURL class provides a way to manipulate URLs and the resources they reference. NSURL objects understand URLs as specified in RFCs 1808, 1738, and 2732. The litmus test for conformance to RFC 1808 is as recommended in RFC 1808—whether the first two characters of `resourceSpecifier` (page 1103) are `@"//"`.

NSURL objects can be used to refer to files, and are the preferred way to do so. ApplicationKit objects that can read data from or write data to a file generally have methods that accept an NSURL object instead of a pathname as the file reference. NSWorkspace provides `openURL:` to open a location specified by a URL. To get the contents of a URL, NSString provides `stringWithContentsOfURL:` and NSData provides `dataWithContentsOfURL:` (page 196).

An NSURL object is composed of two parts—a potentially `nil` base URL and a string that is resolved relative to the base URL. An NSURL object whose string is fully resolved without a base is considered absolute; all others are considered relative.

The NSURL class will fail to create a new NSURL object if the path being passed is not well-formed—the path must comply with RFC 2396. Examples of cases that will not succeed are strings containing space characters and high-bit characters. Should creating an NSURL object fail, the creation methods will return `nil`, which you must be prepared to handle. If you are creating NSURL objects using file system paths, you should use `fileURLWithPath:` (page 1094) or `initFileURLWithPath:` (page 1098), which handle the subtle differences between URL paths and file system paths. If you wish to be tolerant of malformed path strings, you'll need to use functions provided by the Core Foundation framework to clean up the strings.

The informal protocol `NSURLClient` defines a set of methods useful for managing the loading of a URL resource in the background.

See also NSURL Additions in the Application Kit framework, which add methods supporting pasteboards.

NSURL is "toll-free bridged" with its Core Foundation counterpart, CFURL. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object, providing you cast one type to the other. In an API where you see an `NSURL *` parameter, you can pass in a `CFURLRef`, and in an API where you see a `CFURLRef` parameter, you can pass in a pointer to an `NSURL` instance. This approach also applies to your concrete subclasses of NSURL. See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

NSURLHandleClient
- `URLHandleResourceDidBeginLoading:`
- `URLHandleResourceDidCancelLoading:`
- `URLHandleResourceDidFinishLoading:`
- `URLHandle:resourceDataDidBecomeAvailable:`
- `URLHandle:resourceDidFailLoadingWithReason:`

Tasks

Creating an NSURL

- `initWithScheme:host:path:` (page 1099)
  Initializes a newly created NSURL with a specified scheme, host, and path.
+ `URLWithString:` (page 1095)
  Creates and returns an NSURL object initialized with a provided string.
- `initWithString:` (page 1099)
  Initializes an NSURL object with a provided string.
+ `URLWithString:relativeToURL:` (page 1096)
  Creates and returns an NSURL object initialized with a base URL and a relative string.
- `initWithString:relativeToURL:` (page 1100)
  Initializes an NSURL object with a base URL and a relative string.
+ `fileURLWithPath:isDirectory:` (page 1095)
  Initializes and returns a newly created NSURL object as a file URL with a specified path.
+ `fileURLWithPath:` (page 1094)
  Initializes and returns a newly created NSURL object as a file URL with a specified path.
- `initFileURLWithPath:isDirectory:` (page 1098)
  Initializes a newly created NSURL referencing the local file or directory at *path*.
- `initFileURLWithPath:` (page 1098)
  Initializes a newly created NSURL referencing the local file or directory at *path*.

Querying an NSURL

- `isFileURL` (page 1101)
  Returns whether the receiver uses the file scheme.

Accessing the Parts of the URL

- `absoluteString` (page 1096)
  Returns the string for the receiver as if it were an absolute URL.
- `absoluteURL` (page 1097)
  Returns an absolute URL that refers to the same resource as the receiver.
- `baseURL` (page 1097)
  Returns the base URL of the receiver.
- `fragment` (page 1097)
  Returns the fragment of a URL conforming to RFC 1808.
- `host` (page 1097)
  Returns the host of a URL conforming to RFC 1808.
- `parameterString` (page 1101)
  Returns the parameter string of a URL conforming to RFC 1808.

- password (page 1101)
  Returns the password of a URL conforming to RFC 1808.
- path (page 1101)
  Returns the path of a URL conforming to RFC 1808.
- port (page 1102)
  Returns the port number of a URL conforming to RFC 1808.
- query (page 1102)
  Returns the query of a URL conforming to RFC 1808.
- relativePath (page 1102)
  Returns the path of a URL conforming to RFC 1808, without resolving against the receiver's base URL.
- relativeString (page 1103)
  Returns a string representation of the relative portion of the URL.
- resourceSpecifier (page 1103)
  Returns the resource specifier of the URL.
- scheme (page 1103)
  Returns the scheme of the URL.
- standardizedURL (page 1104)
  Returns a new NSURL object with any instances of "../" or "./" removed from its path.
- user (page 1104)
  Returns the user portion of a URL conforming to RFC 1808.

Class Methods

fileURLWithPath:

Initializes and returns a newly created NSURL object as a file URL with a specified path.

```
+ (id)fileURLWithPath:(NSString *)path
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with stringByExpandingTildeInPath (page 1031).

Return Value

An NSURL object initialized with *path*.

Discussion

This method examines *path* in the file system to determine if it is a directory. If *path* is a directory, then a trailing slash is appended. If the file does not exist, it is assumed that *path* represents a directory and a trailing slash is appended. As an alternative, consider using fileURLWithPath:isDirectory: (page 1095) which allows you to explicitly specify whether the returned NSURL object represents a file or directory.

Availability

Available in iPhone OS 2.0 and later.

See Also
initFileURLWithPath: (page 1098)

Declared In
NSURL.h

fileURLWithPath:isDirectory:

Initializes and returns a newly created NSURL object as a file URL with a specified path.

```
+ (id)fileURLWithPath:(NSString *)path
    isDirectory:(BOOL)isDir
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with stringByExpandingTildeInPath (page 1031).

isDir

A Boolean value that specifies whether *path* is treated as a directory path when resolving against relative path components. Pass YES if the *path* indicates a directory, NO otherwise.

Return Value
An NSURL object initialized with *path*.

Availability
Available in iPhone OS 2.0 and later.

See Also
initFileURLWithPath: (page 1098)

Declared In
NSURL.h

URLWithString:

Creates and returns an NSURL object initialized with a provided string.

```
+ (id)URLWithString:(NSString *)URLString
```

Parameters

URLString

The string with which to initialize the NSURL object. Must conform to RFC 2396. This method parses *URLString* according to RFCs 1738 and 1808.

Return Value
An NSURL object initialized with *URLString*. If the string was malformed, returns nil.

Discussion
This method expects *URLString* to contain any necessary percent escape codes, which are ':', '/', '%', '#', ';', and '@'. Note that '%' escapes are translated via UTF-8.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

URLWithString:relativeToURL:

Creates and returns an NSURL object initialized with a base URL and a relative string.

```
+ (id)URLWithString:(NSString *)URLString
    relativeToURL:(NSURL *)baseURL
```

Parameters

URLString
The string with which to initialize the NSURL object. May not be nil. Must conform to RFC 2396. *URLString* is interpreted relative to *baseURL*.

baseURL
The base URL for the NSURL object.

Return Value
An NSURL object initialized with *URLString* and *baseURL*. If *URLString* was malformed, returns nil.

Discussion
This method expects *URLString* to contain any necessary percent escape codes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

Instance Methods

absoluteString

Returns the string for the receiver as if it were an absolute URL.

```
- (NSString *)absoluteString
```

Return Value
An absolute string for the URL. Creating by resolving the receiver's string against its base according to the algorithm given in RFC 1808.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

absoluteURL

Returns an absolute URL that refers to the same resource as the receiver.

```
- (NSURL *)absoluteURL
```

Return Value
An absolute URL that refers to the same resource as the receiver. If the receiver is already absolute, returns `self`. Resolution is performed per RFC 1808.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

baseURL

Returns the base URL of the receiver.

```
- (NSURL *)baseURL
```

Return Value
The base URL of the receiver. If the receiver is an absolute URL, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

fragment

Returns the fragment of a URL conforming to RFC 1808.

```
- (NSString *)fragment
```

Return Value
The fragment of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

host

Returns the host of a URL conforming to RFC 1808.

```
- (NSString *)host
```

Return Value
The host of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

initFileURLWithPath:

Initializes a newly created NSURL referencing the local file or directory at *path*.

```
- (id)initFileURLWithPath:(NSString *)path
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1031).

Return Value
An NSURL object initialized with *path*.

Discussion
Invoking this method is equivalent to invoking `initWithScheme:host:path:` (page 1099) with scheme `NSFileScheme`, a `nil` host, and *path*.

This method examines *path* in the file system to determine if it is a directory. If *path* is a directory, then a trailing slash is appended. If the file does not exist, it is assumed that *path* represents a directory and a trailing slash is appended. As an alternative, consider using `initFileURLWithPath:isDirectory:` (page 1098) which allows you to explicitly specify whether the returned NSURL represents a file or directory.

Availability
Available in iPhone OS 2.0 and later.

See Also
`fileURLWithPath:` (page 1094)

Declared In
NSURL.h

initFileURLWithPath:isDirectory:

Initializes a newly created NSURL referencing the local file or directory at *path*.

```
- (id)initFileURLWithPath:(NSString *)path
    isDirectory:(BOOL)isDir
```

Parameters

path

The path that the NSURL object will represent. *path* should be a valid system path. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1031).

isDir

A Boolean value that specifies whether *path* is treated as a directory path when resolving against relative path components. Pass YES if the *path* indicates a directory, NO otherwise

Return Value
An NSURL object initialized with *path*.

Discussion
Invoking this method is equivalent to invoking initWithScheme:host:path: (page 1099) with scheme NSFileScheme, a nil host, and *path*.

Availability
Available in iPhone OS 2.0 and later.

See Also
fileURLWithPath: (page 1094)

Declared In
NSURL.h

initWithScheme:host:path:

Initializes a newly created NSURL with a specified scheme, host, and path.

```
- (id)initWithScheme:(NSString *)scheme
    host:(NSString *)host
    path:(NSString *)path
```

Parameters

scheme

The scheme for the NSURL object.

host

The host for the NSURL object. May be nil.

path

The path for the NSURL object. If *path* begins with a tilde, it must first be expanded with stringByExpandingTildeInPath (page 1031).

Return Value
The newly initialized NSURL object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

initWithString:

Initializes an NSURL object with a provided string.

```
- (id)initWithString:(NSString *)URLString
```

Parameters

URLString

The string with which to initialize the NSURL object. Must conform to RFC 2396. This method parses *URLString* according to RFCs 1738 and 1808.

Return Value

An NSURL object initialized with *URLString*. If the string was malformed, returns `nil`.

Discussion

This method expects *URLString* to contain any necessary percent escape codes, which are ':', '/', '%', '#', ';', and '@'. Note that '%' escapes are translated via UTF-8.

Availability

Available in iPhone OS 2.0 and later.

See Also

`URLWithString:` (page 1095)

Declared In

`NSURL.h`

initWithString:relativeToURL:

Initializes an NSURL object with a base URL and a relative string.

```
- (id)initWithString:(NSString *)URLString
    relativeToURL:(NSURL *)baseURL
```

Parameters

URLString

The string with which to initialize the NSURL object. Must conform to RFC 2396. *URLString* is interpreted relative to *baseURL*.

baseURL

The base URL for the NSURL object.

Return Value

An NSURL object initialized with *URLString* and *baseURL*. If *URLString* was malformed, returns `nil`.

Discussion

This method expects *URLString* to contain any necessary percent escape codes.

`initWithString:relativeToURL:` is the designated initializer for NSURL.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- baseURL` (page 1097)

`- relativeString` (page 1103)

`URLWithString:relativeToURL:` (page 1096)

Declared In

`NSURL.h`

isFileURL

Returns whether the receiver uses the file scheme.

```
- (BOOL)isFileURL
```

Return Value
Returns `YES` if the receiver uses the file scheme, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

parameterString

Returns the parameter string of a URL conforming to RFC 1808.

```
- (NSString *)parameterString
```

Return Value
The parameter string of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

password

Returns the password of a URL conforming to RFC 1808.

```
- (NSString *)password
```

Return Value
The password of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

path

Returns the path of a URL conforming to RFC 1808.

```
- (NSString *)path
```

Return Value
The path of the URL. If the receiver does not conform to RFC 1808, returns `nil`. If `isFileURL` (page 1101) returns`YES`, the return value is suitable for input into NSFileManager or NSPathUtilities. If the path has a trailing slash it is stripped.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

port

Returns the port number of a URL conforming to RFC 1808.

```
- (NSNumber *)port
```

Return Value
The port number of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

query

Returns the query of a URL conforming to RFC 1808.

```
- (NSString *)query
```

Return Value
The query of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

relativePath

Returns the path of a URL conforming to RFC 1808, without resolving against the receiver's base URL.

```
- (NSString *)relativePath
```

Return Value
The relative path of the URL without resolving against the base URL. If the receiver is an absolute URL, this method returns the same value as `path` (page 1101). If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

relativeString

Returns a string representation of the relative portion of the URL.

```
- (NSString *)relativeString
```

Return Value
A string representation of the relative portion of the URL. If the receiver is an absolute URL this method returns the same value as `absoluteString` (page 1096).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

resourceSpecifier

Returns the resource specifier of the URL.

```
- (NSString *)resourceSpecifier
```

Return Value
The resource specifier of the URL.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

scheme

Returns the scheme of the URL.

```
- (NSString *)scheme
```

Return Value
The scheme of the URL.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURL.h

standardizedURL

Returns a new NSURL object with any instances of "." or ".." removed from its path.

```
- (NSURL *)standardizedURL
```

Return Value
A new `NSURL` object initialized with a version of the receiver's URL that has had any instances of "." or ".." removed from its path.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

user

Returns the user portion of a URL conforming to RFC 1808.

```
- (NSString *)user
```

Return Value
The user portion of the URL. If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURL.h`

Constants

NSURL Schemes

These schemes are the ones that `NSURL` can parse.

```
extern NSString *NSURLFileScheme;
```

Constants
`NSURLFileScheme`

Identifies a URL that points to a file on a mounted volume.

Available in iPhone OS 2.0 and later.

Declared in `NSURL.h`

Discussion
For more information, see initWithScheme:host:path: (page 1099).

Declared In
`NSURL.h`

NSURLAuthenticationChallenge Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLAuthenticationChallenge.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSURLAuthenticationChallenge encapsulates a challenge from a server requiring authentication from the client.

Tasks

Creating an Authentication Challenge Instance

- `initWithAuthenticationChallenge:sender:` (page 1107)
  Returns an initialized NSURLAuthenticationChallenge object copying the properties from *challenge*, and setting the authentication sender to *sender*.

- `initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender:` (page 1107)
  Returns an initialized NSURLAuthenticationChallenge object for the specified *space* using the *credential*, or `nil` if there is no proposed credential.

Getting Authentication Challenge Properties

- `error` (page 1106)
  Returns the NSError object representing the last authentication failure.
- `failureResponse` (page 1107)
  Returns the NSURLResponse object representing the last authentication failure.
- `previousFailureCount` (page 1108)
  Returns the receiver's count of failed authentication attempts.
- `proposedCredential` (page 1108)
  Returns the proposed credential for this challenge.
- `protectionSpace` (page 1108)
  Returns the receiver's protection space.
- `sender` (page 1109)
  Returns the receiver's sender.

Instance Methods

error

Returns the NSError object representing the last authentication failure.

```
- (NSError *)error
```

Discussion

This method returns `nil` if the protocol doesn't use errors to indicate an authentication failure.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `failureResponse` (page 1107)

Declared In
NSURLAuthenticationChallenge.h

failureResponse

Returns the NSURLResponse object representing the last authentication failure.

```
- (NSURLResponse *)failureResponse
```

Discussion
This method will return `nil` if the protocol doesn't use responses to indicate an authentication failure.

Availability
Available in iPhone OS 2.0 and later.

See Also
- error (page 1106)

Declared In
NSURLAuthenticationChallenge.h

initWithAuthenticationChallenge:sender:

Returns an initialized NSURLAuthenticationChallenge object copying the properties from *challenge*, and setting the authentication sender to *sender*.

```
- (id)initWithAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
    sender:(id < NSURLAuthenticationChallengeSender >)sender
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender: (page 1107)

Declared In
NSURLAuthenticationChallenge.h

initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender:

Returns an initialized NSURLAuthenticationChallenge object for the specified *space* using the *credential*, or `nil` if there is no proposed credential.

```
- (id)initWithProtectionSpace:(NSURLProtectionSpace *)space
    proposedCredential:(NSURLCredential *)credential
    previousFailureCount:(NSInteger)count failureResponse:(NSURLResponse *)response
     error:(NSError *)error sender:(id < NSURLAuthenticationChallengeSender >)sender
```

Discussion
The previous failure count is set to *count*. The *response* should contain the NSURLResponse for the authentication failure, or `nil` if it is not applicable to the challenge. The *error* should contain the NSError for the authentication failure, or `nil` if it is not applicable to the challenge. The object that initiated the authentication challenge is set to *sender*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithAuthenticationChallenge:sender:` (page 1107)

Declared In
`NSURLAuthenticationChallenge.h`

previousFailureCount

Returns the receiver's count of failed authentication attempts.

```
- (NSInteger)previousFailureCount
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLAuthenticationChallenge.h`

proposedCredential

Returns the proposed credential for this challenge.

```
- (NSURLCredential *)proposedCredential
```

Discussion
This method will return `nil` if there is no default credential for this challenge.

If the proposed credential is not `nil` and returns `YES` when sent the message `hasPassword` (page 1135), then the credential is ready to use as-is. If the proposed credential returns `NO` for `hasPassword`, then the credential provides a default user name and the client must prompt the user for a corresponding password.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLAuthenticationChallenge.h`

protectionSpace

Returns the receiver's protection space.

```
- (NSURLProtectionSpace *)protectionSpace
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLAuthenticationChallenge.h

sender

Returns the receiver's sender.

```
- (id < NSURLAuthenticationChallengeSender >)sender
```

Discussion
The sender should be sent a `useCredential:forAuthenticationChallenge:` (page 1317), `continueWithoutCredentialForAuthenticationChallenge:` (page 1316) or `cancelAuthenticationChallenge:` (page 1316) when the client is finished processing the authentication challenge.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLAuthenticationChallenge.h

NSURLCache Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLCache.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSURLCache implements the caching of responses to URL load requests by mapping NSURLRequest objects to NSCachedURLResponse objects. It is a composite of an in-memory and an on-disk cache.

Methods are provided to manipulate the sizes of each of these caches as well as to control the path on disk to use for persistent storage of cache data.

Tasks

Getting and Setting Shared Cache

+ `sharedURLCache` (page 1113)
  Returns the shared NSURLCache instance.

+ `setSharedURLCache:` (page 1113)
  Sets the shared NSURLCache instance to a specified cache object.

Creating a New Cache Object

- `initWithMemoryCapacity:diskCapacity:diskPath:` (page 1115)
  Initializes an NSURLCache object with the specified values.

Getting and Storing Cached Objects

- `cachedResponseForRequest:` (page 1114)
  Returns the cached URL response in the cache for the specified URL request.

- `storeCachedResponse:forRequest:` (page 1118)
  Stores a cached URL response for a specified request

Removing Cached Objects

- `removeAllCachedResponses` (page 1116)
  Clears the receiver's cache, removing all stored cached URL responses.

- `removeCachedResponseForRequest:` (page 1117)
  Removes the cached URL response for a specified URL request.

Getting and Setting On-disk Cache Properties

- `currentDiskUsage` (page 1114)
  Returns the current size of the receiver's on-disk cache, in bytes.

- `diskCapacity` (page 1115)
  Returns the capacity of the receiver's on-disk cache, in bytes.

- `setDiskCapacity:` (page 1117)
  Sets the receiver's on-disk cache capacity

Getting and Setting In-memory Cache Properties

- `currentMemoryUsage` (page 1115)
  Returns the current size of the receiver's in-memory cache, in bytes.

- memoryCapacity (page 1116)
  Returns the capacity of the receiver's in-memory cache, in bytes.
- setMemoryCapacity: (page 1117)
  Sets the receiver's in-memory cache capacity.

Class Methods

setSharedURLCache:

Sets the shared NSURLCache instance to a specified cache object.

```
+ (void)setSharedURLCache:(NSURLCache *)cache
```

Parameters

cache
The cache object to use as the shared cache object.

Discussion

Applications that have special caching requirements or constraints should use this method to specify an NSURLCache instance with customized cache settings.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ sharedURLCache (page 1113)

Declared In

NSURLCache.h

sharedURLCache

Returns the shared NSURLCache instance.

```
+ (NSURLCache *)sharedURLCache
```

Return Value

The shared NSURLCache instance.

Discussion

The disk path is set to: <user_home_directory>/Library/Caches/<current_process_name>. The user's home directory is determined by calling NSHomeDirectory (page 1360) and the current process name is determined using [[NSProcessInfo processInfo] processName].

Applications that do not have special caching requirements or constraints should find the default shared cache instance acceptable. Applications with more specific needs can create a custom NSURLCache object and set it as the shared cache instance using setSharedURLCache: (page 1113).

Availability

Available in iPhone OS 2.0 and later.

See Also
+ setSharedURLCache: (page 1113)

Declared In
NSURLCache.h

Instance Methods

cachedResponseForRequest:

Returns the cached URL response in the cache for the specified URL request.

```
- (NSCachedURLResponse *)cachedResponseForRequest:(NSURLRequest *)request
```

Parameters

request

The URL request whose cached response is desired.

Return Value
The cached URL response for *request*, or nil if no response has been cached.

Availability
Available in iPhone OS 2.0 and later.

See Also
- storeCachedResponse:forRequest: (page 1118)

Declared In
NSURLCache.h

currentDiskUsage

Returns the current size of the receiver's on-disk cache, in bytes.

```
- (NSUInteger)currentDiskUsage
```

Return Value
The current size of the receiver's on-disk cache, in bytes.

Availability
Available in iPhone OS 2.0 and later.

See Also
- diskCapacity (page 1115)
- setDiskCapacity: (page 1117)

Declared In
NSURLCache.h

currentMemoryUsage

Returns the current size of the receiver's in-memory cache, in bytes.

```
- (NSUInteger)currentMemoryUsage
```

Return Value

The current size of the receiver's in-memory cache, in bytes.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `memoryCapacity` (page 1116)
- `setMemoryCapacity:` (page 1117)

Declared In

`NSURLCache.h`

diskCapacity

Returns the capacity of the receiver's on-disk cache, in bytes.

```
- (NSUInteger)diskCapacity
```

Return Value

The capacity of the receiver's on-disk cache, in bytes.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `currentDiskUsage` (page 1114)
- `setDiskCapacity:` (page 1117)

Declared In

`NSURLCache.h`

initWithMemoryCapacity:diskCapacity:diskPath:

Initializes an NSURLCache object with the specified values.

```
- (id)initWithMemoryCapacity:(NSUInteger)memoryCapacity
    diskCapacity:(NSUInteger)diskCapacity diskPath:(NSString *)path
```

Parameters

memoryCapacity

The memory capacity of the cache, in bytes.

diskCapacity

The disk capacity of the cache, in bytes.

path

The location at which to store the on-disk cache.

Return Value
The initialized NSURLCache object.

Discussion
The returned NSURLCache is backed by disk, so developers can be more liberal with space when choosing the capacity for this kind of cache. A disk cache measured in the tens of megabytes should be acceptable in most cases.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ sharedURLCache` (page 1113)

Declared In
`NSURLCache.h`

memoryCapacity

Returns the capacity of the receiver's in-memory cache, in bytes.

```
- (NSUInteger)memoryCapacity
```

Return Value
The capacity of the receiver's in-memory cache, in bytes.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- currentMemoryUsage` (page 1115)
`- setMemoryCapacity:` (page 1117)

Declared In
`NSURLCache.h`

removeAllCachedResponses

Clears the receiver's cache, removing all stored cached URL responses.

```
- (void)removeAllCachedResponses
```

Availability
Available in iPhone OS 2.0 and later.

See Also
`- removeCachedResponseForRequest:` (page 1117)

Declared In
`NSURLCache.h`

removeCachedResponseForRequest:

Removes the cached URL response for a specified URL request.

```
- (void)removeCachedResponseForRequest:(NSURLRequest *)request
```

Parameters

request

The URL request whose cached URL response should be removed. If there is no corresponding cached URL response, no action is taken.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeAllCachedResponses (page 1116)

Declared In

NSURLCache.h

setDiskCapacity:

Sets the receiver's on-disk cache capacity

```
- (void)setDiskCapacity:(NSUInteger)diskCapacity
```

Parameters

diskCapacity

The new on-disk cache capacity, in bytes. The on-disk cache will truncate its contents to *diskCapacity*, if necessary.

Availability

Available in iPhone OS 2.0 and later.

See Also

- currentDiskUsage (page 1114)
- diskCapacity (page 1115)

Declared In

NSURLCache.h

setMemoryCapacity:

Sets the receiver's in-memory cache capacity.

```
- (void)setMemoryCapacity:(NSUInteger)memoryCapacity
```

Parameters

memoryCapacity

The new in-memory cache capacity, in bytes. The in-memory cache will truncate its contents to *memoryCapacity*, if necessary.

Availability

Available in iPhone OS 2.0 and later.

See Also
- `currentMemoryUsage` (page 1115)
- `memoryCapacity` (page 1116)

Declared In
`NSURLCache.h`

storeCachedResponse:forRequest:

Stores a cached URL response for a specified request

```
- (void)storeCachedResponse:(NSCachedURLResponse *)cachedResponse
    forRequest:(NSURLRequest *)request
```

Parameters

cachedResponse
The cached URL response to store.

request
The request for which the cached URL response is being stored.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `cachedResponseForRequest:` (page 1114)

Declared In
`NSURLCache.h`

NSURLConnection Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLConnection.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An NSURLConnection object provides support to perform the loading of a URL request. The interface for NSURLConnection is sparse, providing only the controls to start and cancel asynchronous loads of a URL request.

NSURLConnection's delegate methods allow an object to receive informational callbacks about the asynchronous load of a URL request. Other delegate methods provide facilities that allow the delegate to customize the process of performing an asynchronous URL load.

Note that these delegate methods will be called on the thread that started the asynchronous load operation for the associated NSURLConnection object.

The following contract governs the delegate methods defined in this interface:

- Zero or more `connection:willSendRequest:redirectResponse:` (page 1130) messages will be sent to the delegate before any further messages are sent if it is determined that the download must redirect to a new location. The delegate can allow the redirect, modify the destination or deny the redirect.
- Zero or more `connection:didReceiveAuthenticationChallenge:` (page 1128) messages will be sent to the delegate if it is necessary to authenticate in order to download the request and NSURLConnection does not already have authenticated credentials.
- Zero or more `connection:didCancelAuthenticationChallenge:` (page 1127) messages will be sent to the delegate if the connection cancels the authentication challenge due to the protocol implementation encountering an error.
- Zero or more `connection:didReceiveResponse:` (page 1129) messages will be sent to the delegate before receiving a `connection:didReceiveData:` (page 1129) message. The only case where `connection:didReceiveResponse:` is not sent to a delegate is when the protocol implementation encounters an error before a response could be created.
- Zero or more `connection:didReceiveData:` (page 1129) messages will be sent before any of the following messages are sent to the delegate: `connection:willCacheResponse:` (page 1130), `connectionDidFinishLoading:` (page 1131), `connection:didFailWithError:` (page 1127).
- Zero or one `connection:willCacheResponse:` (page 1130) messages will be sent to the delegate after `connection:didReceiveData:` (page 1129) is sent but before a `connectionDidFinishLoading:` (page 1131) message is sent.
- Unless a NSURLConnection receives a `cancel` (page 1124) message, the delegate will receive one and only one of `connectionDidFinishLoading:` (page 1131), or `connection:didFailWithError:` (page 1127) message, but never both. In addition, once either of messages are sent, the delegate will receive no further messages for the given NSURLConnection.

NSURLConnection also has a convenience class method, `sendSynchronousRequest:returningResponse:error:` (page 1123), to load a URL request synchronously.

`NSHTTPURLResponse` is a subclass of `NSURLResponse` that provides methods for accessing information specific to HTTP protocol responses. An `NSHTTPURLResponse` object represents a response to an HTTP URL load request.

Tasks

Preflighting a Request

`+ canHandleRequest:` (page 1122)
Returns whether a request can be handled based on a "preflight" evaluation.

Loading Data Synchronously

`+ sendSynchronousRequest:returningResponse:error:` (page 1123)
Performs a synchronous load of the specified URL request.

Loading Data Asynchronously

+ `connectionWithRequest:delegate:` (page 1122)
  Creates and returns an initialized URL connection and begins to load the data for the URL request.

- `initWithRequest:delegate:` (page 1124)
  Returns an initialized URL connection and begins to load the data for the URL request.

- `initWithRequest:delegate:startImmediately:` (page 1125)
  Returns an initialized URL connection and begins to load the data for the URL request, if specified.

- `start` (page 1126)
  Causes the receiver to begin loading data, if it has not already.

Stopping a Connection

- `cancel` (page 1124)
  Cancels an asynchronous load of a request.

Runloop Scheduling

- `scheduleInRunLoop:forMode:` (page 1125)
  Determines the runloop and mode that the receiver uses to send delegate messages to the receiver.

- `unscheduleFromRunLoop:forMode:` (page 1126)
  Causes the receiver to stop sending delegate messages using the specified runloop and mode.

Connection Authentication

- `connection:didCancelAuthenticationChallenge:` (page 1127) *delegate method*
  Sent when a connection cancels an authentication challenge.

- `connection:didReceiveAuthenticationChallenge:` (page 1128) *delegate method*
  Sent when a connection must authenticate a challenge in order to download its request.

Connection Data and Responses

- `connection:willCacheResponse:` (page 1130) *delegate method*
  Sent before the connection stores a cached response in the cache, to give the delegate an opportunity to alter it.

- `connection:didReceiveResponse:` (page 1129) *delegate method*
  Sent when the connection has received sufficient data to construct the URL response for its request.

- `connection:didReceiveData:` (page 1129) *delegate method*
  Sent as a connection loads data incrementally.

- connection:willSendRequest:redirectResponse: (page 1130) *delegate method*
  Sent when the connection determines that it must change URLs in order to continue loading a request.

Connection Completion

- connection:didFailWithError: (page 1127) *delegate method*
  Sent when a connection fails to load its request successfully.
- connectionDidFinishLoading: (page 1131) *delegate method*
  Sent when a connection has finished loading successfully.

Class Methods

canHandleRequest:

Returns whether a request can be handled based on a "preflight" evaluation.

```
+ (BOOL)canHandleRequest:(NSURLRequest *)request
```

Parameters

request
  The request to evaluate.

Return Value

YES if a “preflight” operation determines that a connection with *request* can be created and the associated I/O can be started, NO otherwise.

Discussion

The result of this method is valid as long as no NSURLProtocol classes are registered or unregistered, and the specified *request* remains unchanged. Applications should be prepared to handle failures even if they have performed request preflighting by calling this method.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ registerClass: (page 1157)
+ unregisterClass: (page 1159)

Declared In

NSURLConnection.h

connectionWithRequest:delegate:

Creates and returns an initialized URL connection and begins to load the data for the URL request.

```
+ (NSURLConnection *)connectionWithRequest:(NSURLRequest *)request
    delegate:(id)delegate
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate

The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. For the connection to work correctly the calling thread's run loop must be operating in the default run loop mode.]

Return Value

The URL connection for the URL request. Returns `nil` if a connection can't be created.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithRequest:delegate:` (page 1124)

Declared In

`NSURLConnection.h`

sendSynchronousRequest:returningResponse:error:

Performs a synchronous load of the specified URL request.

```
+ (NSData *)sendSynchronousRequest:(NSURLRequest *)request
    returningResponse:(NSURLResponse **)response error:(NSError **)error
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

response

Out parameter for the URL response returned by the server.

error

Out parameter used if an error occurs while processing the request. May be `NULL`.

Return Value

The downloaded data for the URL request. Returns `nil` if a connection could not be created or if the download fails.

Discussion

A synchronous load is built on top of the asynchronous loading code made available by the class. The calling thread is blocked while the asynchronous loading system performs the URL load on a thread spawned specifically for this load request. No special threading or run loop configuration is necessary in the calling thread in order to perform a synchronous load.

If authentication is required in order to download the request, the required credentials must be specified as part of the URL. If authentication fails, or credentials are missing, the connection will attempt to continue without credentials.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

Instance Methods

cancel

Cancels an asynchronous load of a request.

```
- (void)cancel
```

Discussion
Once this method is called, the receiver's delegate will no longer receive any messages for this NSURLConnection.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ connectionWithRequest:delegate: (page 1122)
- initWithRequest:delegate: (page 1124)

Declared In
NSURLConnection.h

initWithRequest:delegate:

Returns an initialized URL connection and begins to load the data for the URL request.

```
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate
```

Parameters

request
The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate
The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. By default, for the connection to work correctly the calling thread's run loop must be operating in the default run loop mode. See scheduleInRunLoop:forMode: (page 1125) to change the runloop and mode.

Return Value
The URL connection for the URL request. Returns nil if a connection can't be initialized.

Availability
Available in iPhone OS 2.0 and later.

See Also

+ connectionWithRequest:delegate: (page 1122)
- initWithRequest:delegate:startImmediately: (page 1125)

Declared In

NSURLConnection.h

initWithRequest:delegate:startImmediately:

Returns an initialized URL connection and begins to load the data for the URL request, if specified.

```
- (id)initWithRequest:(NSURLRequest *)request delegate:(id)delegate
    startImmediately:(BOOL)startImmediately
```

Parameters

request

The URL request to load. The *request* object is deep-copied as part of the initialization process. Changes made to *request* after this method returns do not affect the request that is used for the loading process.

delegate

The delegate object for the connection. The delegate will receive delegate messages as the load progresses. Messages to the delegate will be sent on the thread that calls this method. By default, for the connection to work correctly the calling thread's run loop must be operating in the default run loop mode. See scheduleInRunLoop:forMode: (page 1125) to change the runloop and mode.]

startImmediately

YES if the connection should being loading data immediately, otherwise NO.

Return Value

The URL connection for the URL request. Returns nil if a connection can't be initialized.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLConnection.h

scheduleInRunLoop:forMode:

Determines the runloop and mode that the receiver uses to send delegate messages to the receiver.

```
- (void)scheduleInRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters

aRunLoop

The NSRunloop instance to use for delegate messages.

mode

The mode in which to supply delegate messages.

Discussion
At creation, a connection is scheduled on the current thread (the one where the creation takes place) in the default mode. That can be changed to add or remove runloop + mode pairs using the following methods. It is permissible to be scheduled on multiple run loops and modes, or on the same run loop in multiple modes, so scheduling in one place does not cause unscheduling in another.

You may call these methods after the connection has started. However, if the connection is scheduled on multiple threads or if you are not calling these methods from the thread where the connection is scheduled, there is a race between these methods and the delivery of delegate methods on the other threads. The caller must either be prepared for additional delegation messages on the other threads, or must halt the run loops on the other threads before calling these methods to guarantee that no further callbacks will occur.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

start
Causes the receiver to begin loading data, if it has not already.

```
- (void)start
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

unscheduleFromRunLoop:forMode:
Causes the receiver to stop sending delegate messages using the specified runloop and mode.

```
- (void)unscheduleFromRunLoop:(NSRunLoop *)aRunLoop forMode:(NSString *)mode
```

Parameters
aRunLoop
The runloop instance to unschedule.

mode
The mode to unschedule.

Discussion
At creation, a connection is scheduled on the current thread (the one where the creation takes place) in the default mode. That can be changed to add or remove runloop + mode pairs using the following methods. It is permissible to be scheduled on multiple run loops and modes, or on the same run loop in multiple modes, so scheduling in one place does not cause unscheduling in another.

You may call these methods after the connection has started. However, if the connection is scheduled on multiple threads or if you are not calling these methods from the thread where the connection is scheduled, there is a race between these methods and the delivery of delegate methods on the other

threads. The caller must either be prepared for additional delegation messages on the other threads, or must halt the run loops on the other threads before calling these methods to guarantee that no further callbacks will occur.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

Delegate Methods

connection:didCancelAuthenticationChallenge:

Sent when a connection cancels an authentication challenge.

```
- (void)connection:(NSURLConnection *)connection
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

connection
The connection sending the message.

challenge
The challenge that was canceled.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

connection:didFailWithError:

Sent when a connection fails to load its request successfully.

```
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error
```

Parameters

connection
The connection sending the message.

error
An error object containing details of why the connection failed to load the request successfully.

Discussion
Once the delegate receives this message, it will receive no further messages for *connection*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLConnection.h

connection:didReceiveAuthenticationChallenge:

Sent when a connection must authenticate a challenge in order to download its request.

```
- (void)connection:(NSURLConnection *)connection
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

connection

The connection sending the message.

challenge

The challenge that *connection* must authenticate in order to download its request.

Discussion

This method gives the delegate the opportunity to determine the course of action taken for the challenge: provide credentials, continue without providing credentials, or cancel the authentication challenge and the download.

The delegate can determine the number of previous authentication challenges by sending the message `previousFailureCount` (page 1108) to *challenge*.

If the previous failure count is 0 and the value returned by `proposedCredential` (page 1108) is `nil`, the delegate can create a new NSURLCredential object, providing a user name and password, and send a `useCredential:forAuthenticationChallenge:` (page 1317) message to `[challenge sender]`, passing the credential and *challenge* as parameters. If `proposedCredential` is not `nil`, the value is a credential from the URL or the shared credential storage that can be provided to the user as feedback.

The delegate may decide to abandon further attempts at authentication at any time by sending `[challenge sender]` a `continueWithoutCredentialForAuthenticationChallenge:` (page 1316) or a `cancelAuthenticationChallenge:` (page 1316) message. The specific action will be implementation dependent.

If the delegate implements this method, the download will suspend until `[challenge sender]` is sent one of the following messages: `useCredential:forAuthenticationChallenge:` (page 1317), `continueWithoutCredentialForAuthenticationChallenge:` (page 1316) or `cancelAuthenticationChallenge:` (page 1316).

If the delegate does not implement this method the default implementation is used. If a valid credential for the request is provided as part of the URL, or is available from the NSURLCredentialStorage the `[challenge sender]` is sent a `useCredential:forAuthenticationChallenge:` (page 1317) with the credential. If the challenge has no credential or the credentials fail to authorize access, then `continueWithoutCredentialForAuthenticationChallenge:` (page 1316) is sent to `[challenge sender]` instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `cancelAuthenticationChallenge:` (page 1316)
- `continueWithoutCredentialForAuthenticationChallenge:` (page 1316)
- `useCredential:forAuthenticationChallenge:` (page 1317)

Declared In

`NSURLConnection.h`

connection:didReceiveData:

Sent as a connection loads data incrementally.

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
```

Parameters

connection

The connection sending the message.

data

The newly available data. The delegate should concatenate the contents of each *data* object delivered to build up the complete data for a URL load.

Discussion

This method provides the only way for an asynchronous delegate to retrieve the loaded data. It is the responsibility of the delegate to retain or copy this data as it is delivered.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLConnection.h

connection:didReceiveResponse:

Sent when the connection has received sufficient data to construct the URL response for its request.

```
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse
    *)response
```

Parameters

connection

The connection sending the message.

response

The URL response for the connection's request. This object is immutable and will not be modified by the URL loading system once it is presented to the delegate.

Discussion

In rare cases, for example in the case of an HTTP load where the content type of the load data is `multipart/x-mixed-replace`, the delegate will receive more than one `connection:didReceiveResponse:` message. In the event this occurs, delegates should discard all data previously delivered by `connection:didReceiveData:`, and should be prepared to handle the, potentially different, MIME type reported by the newly reported URL response.

The only case where this message is not sent to the delegate is when the protocol implementation encounters an error before a response could be created.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLConnection.h

connection:willCacheResponse:

Sent before the connection stores a cached response in the cache, to give the delegate an opportunity to alter it.

```
- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
    willCacheResponse:(NSCachedURLResponse *)cachedResponse
```

Parameters

connection

The connection sending the message.

cachedResponse

The proposed cached response to store in the cache.

Return Value

The actual cached response to store in the cache. The delegate may return *cachedResponse* unmodified, return a modified cached response, or return `nil` if no cached response should be stored for the connection.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLConnection.h`

connection:willSendRequest:redirectResponse:

Sent when the connection determines that it must change URLs in order to continue loading a request.

```
- (NSURLRequest *)connection:(NSURLConnection *)connection
    willSendRequest:(NSURLRequest *)request redirectResponse:(NSURLResponse
    *)redirectResponse
```

Parameters

connection

The connection sending the message.

request

The proposed redirected request. The delegate should inspect the redirected request to verify that it meets its needs, and create a copy with new attributes to return to the connection if necessary.

redirectResponse

The URL response that caused the redirect. May be *nil* in cases where this method is not being sent as a result of involving the delegate in redirect processing.

Return Value

The actual URL request to use in light of the redirection response. The delegate may copy and modify *request* as necessary to change its attributes, return *request* unmodified, or return `nil`.

Discussion

If the delegate wishes to cancel the redirect, it should call the *connection* object's `cancel` method. Alternatively, the delegate method can return `nil` to cancel the redirect, and the connection will continue to process. This has special relevance in the case where *redirectResponse* is not `nil`. In

this case, any data that is loaded for the connection will be sent to the delegate, and the delegate will receive a `connectionDidFinishLoading` or `connection:didFailLoadingWithError:` message, as appropriate.

Special Considerations

The delegate can receive this message as a result of transforming a request's URL to its canonical form, or for protocol-specific reasons, such as an HTTP redirect. The delegate implementation should be prepared to receive this message multiple times.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLConnection.h`

connectionDidFinishLoading:

Sent when a connection has finished loading successfully.

```
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
```

Parameters

connection

The connection sending the message.

Discussion

The delegate will receive no further messages for *connection*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLConnection.h`

NSURLCredential Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLCredential.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSURLCredential` is an immutable object representing an authentication credential consisting of the user name, a password and the type of persistent storage to use, if any.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 1250)

Tasks

Creating a Credential

`+ credentialWithUser:password:persistence:` (page 1134)
Creates and returns an `NSURLCredential` object with a given user name and password using a given persistence setting.

`- initWithUser:password:persistence:` (page 1135)
Returns an `NSURLCredential` object initialized with a given user name and password using a given persistence setting.

Getting Credential Properties

`- hasPassword` (page 1135)
Returns a Boolean value that indicates whether the receiver has a password.

`- password` (page 1136)
Returns the receiver's password.

`- persistence` (page 1136)
Returns the receiver's persistence setting.

`- user` (page 1137)
Returns the receiver's user name.

Class Methods

credentialWithUser:password:persistence:

Creates and returns an `NSURLCredential` object with a given user name and password using a given persistence setting.

```
+ (NSURLCredential *)credentialWithUser:(NSString *)user password:(NSString
    *)password persistence:(NSURLCredentialPersistence)persistence
```

Parameters

user
The user for the credential.

password
The password for *user*.

persistence
The persistence setting for the credential.

Return Value

An `NSURLCredential` object with user name *user*, password *password*, and using persistence setting *persistence*.

Discussion
If *persistence* is `NSURLCredentialPersistencePermanent` the credential is stored in the keychain.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithUser:password:persistence:` (page 1135)

Declared In
`NSURLCredential.h`

Instance Methods

hasPassword

Returns a Boolean value that indicates whether the receiver has a password.

```
- (BOOL)hasPassword
```

Return Value
`YES` if the receiver has a password, `NO` otherwise.

Discussion
This method does not attempt to retrieve the password.

If this credential's password is stored in the user's keychain, `password` (page 1136) may return `NO` even if this method returns `YES`, since getting the password may fail, or the user may refuse access.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCredential.h`

initWithUser:password:persistence:

Returns an `NSURLCredential` object initialized with a given user name and password using a given persistence setting.

```
- (id)initWithUser:(NSString *)user password:(NSString *)password
    persistence:(NSURLCredentialPersistence)persistence
```

Parameters

user
The user for the credential.

password
The password for *user*.

persistence
The persistence setting for the credential.

Return Value
An `NSURLCredential` object initialized with user name *`user`*, password *`password`*, and using persistence setting *`persistence`*.

Discussion
If *`persistence`* is `NSURLCredentialPersistencePermanent` the credential is stored in the keychain.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ credentialWithUser:password:persistence:` (page 1134)

Declared In
`NSURLCredential.h`

password

Returns the receiver's password.

```
- (NSString *)password
```

Return Value
The receiver's password.

Discussion
If the password is stored in the user's keychain, this method may result in prompting the user for access.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- hasPassword` (page 1135)

Declared In
`NSURLCredential.h`

persistence

Returns the receiver's persistence setting.

```
- (NSURLCredentialPersistence)persistence
```

Return Value
The receiver's persistence setting.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCredential.h`

user

Returns the receiver's user name.

```
- (NSString *)user
```

Return Value
The receiver's user name.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLCredential.h

Constants

NSURLCredentialPersistence

These constants specify how long the credential will be kept.

```
typedef enum {
    NSURLCredentialPersistenceNone,
    NSURLCredentialPersistenceForSession,
    NSURLCredentialPersistencePermanent
} NSURLCredentialPersistence;
```

Constants

`NSURLCredentialPersistenceNone`
Credential won't be stored.
Available in iPhone OS 2.0 and later.
Declared in `NSURLCredential.h`

`NSURLCredentialPersistenceForSession`
Credential will be stored only for this session.
Available in iPhone OS 2.0 and later.
Declared in `NSURLCredential.h`

`NSURLCredentialPersistencePermanent`
Credential will be stored in the user's keychain and shared with other applications.
Available in iPhone OS 2.0 and later.
Declared in `NSURLCredential.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLCredential.h

NSURLCredentialStorage Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLCredentialStorage.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSURLCredentialStorage implements a singleton (shared object) that manages the credential storage.

Tasks

Getting the Credential Storage

`+ sharedCredentialStorage` (page 1140)
Returns the shared URL credential storage object.

Getting and Setting Default Credentials

- `defaultCredentialForProtectionSpace:` (page 1141)
  Returns the default credential for the specified *protectionSpace*.
- `setDefaultCredential:forProtectionSpace:` (page 1143)
  Sets the default credential for a specified protection space.

Adding and Removing Credentials

- `removeCredential:forProtectionSpace:` (page 1142)
  Removes a specified credential from the credential storage for the specified protection space.
- `setCredential:forProtectionSpace:` (page 1142)
  Adds *credential* to the credential storage for the specified *protectionSpace*.

Retrieving Credentials

- `allCredentials` (page 1140)
  Returns a dictionary containing the credentials for all available protection spaces.
- `credentialsForProtectionSpace:` (page 1141)
  Returns a dictionary containing the credentials for the specified protection space.

Class Methods

sharedCredentialStorage

Returns the shared URL credential storage object.

```
+ (NSURLCredentialStorage *)sharedCredentialStorage
```

Return Value
The shared NSURLCredentialStorage object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLCredentialStorage.h`

Instance Methods

allCredentials

Returns a dictionary containing the credentials for all available protection spaces.

```
- (NSDictionary *)allCredentials
```

Return Value
A dictionary containing the credentials for all available protection spaces. The dictionary has keys corresponding to the `NSURLProtectionSpace` objects. The values for the `NSURLProtectionSpace` keys consist of dictionaries where the keys are user name strings, and the value is the corresponding `NSURLCredential` object.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- credentialsForProtectionSpace:` (page 1141)

Declared In
`NSURLCredentialStorage.h`

credentialsForProtectionSpace:

Returns a dictionary containing the credentials for the specified protection space.

```
- (NSDictionary *)credentialsForProtectionSpace:(NSURLProtectionSpace
    *)protectionSpace
```

Parameters
protectionSpace
The protection space whose credentials you want to retrieve.

Return Value
A dictionary containing the credentials for *protectionSpace*. The dictionary's keys are user name strings, and the value is the corresponding `NSURLCredential`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- allCredentials` (page 1140)

Declared In
`NSURLCredentialStorage.h`

defaultCredentialForProtectionSpace:

Returns the default credential for the specified *protectionSpace*.

```
- (NSURLCredential *)defaultCredentialForProtectionSpace:(NSURLProtectionSpace
    *)protectionSpace
```

Parameters
protectionSpace
The URL protection space of interest.

Return Value
The default credential for *protectionSpace* or `nil` if no default has been set.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setDefaultCredential:forProtectionSpace:` (page 1143)

Declared In
`NSURLCredentialStorage.h`

removeCredential:forProtectionSpace:

Removes a specified credential from the credential storage for the specified protection space.

```
- (void)removeCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential
The credential to remove.

protectionSpace
The protection space from which to remove the credential.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `setCredential:forProtectionSpace:` (page 1142)

Declared In
`NSURLCredentialStorage.h`

setCredential:forProtectionSpace:

Adds *credential* to the credential storage for the specified *protectionSpace*.

```
- (void)setCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential
The credential to add. If a credential with the same user name already exists in *protectionSpace*, then *credential* replaces the existing object.

protectionSpace
The protection space to which to add the credential.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeCredential:forProtectionSpace:` (page 1142)

Declared In
`NSURLCredentialStorage.h`

setDefaultCredential:forProtectionSpace:

Sets the default credential for a specified protection space.

```
- (void)setDefaultCredential:(NSURLCredential *)credential
    forProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

credential

The URL credential to set as the default for *protectionSpace*. If the receiver does not contain *credential* in the specified *protectionSpace* it will be added.

protectionSpace

The protection space whose default credential is being set.

Availability

Available in iPhone OS 2.0 and later.

See Also

- defaultCredentialForProtectionSpace: (page 1141)

Declared In

NSURLCredentialStorage.h

Notifications

NSURLCredentialStorageChangedNotification

This notification is posted when the set of stored credentials changes.

The notification object is the NSURLCredentialStorage instance. This notification does not contain a userInfo dictionary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLCredentialStorage.h

NSURLProtectionSpace Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLProtectionSpace.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

NSURLProtectionSpace represents a server or an area on a server, commonly referred to as a realm, that requires authentication. An NSURLProtectionSpace's credentials apply to any requests within that protection space.

Adopted Protocols

NSCopying

- copyWithZone: (page 1250)

Tasks

Creating a Protection Space

- initWithHost:port:protocol:realm:authenticationMethod: (page 1147)
  Initializes a protection space object.
- initWithProxyHost:port:type:realm:authenticationMethod: (page 1148)
  Initializes a protection space object representing a proxy server.

Getting Protection Space Properties

- authenticationMethod (page 1146)
  Returns the authentication method used by the receiver.
- host (page 1147)
  Returns the receiver's host.
- isProxy (page 1148)
  Returns whether the receiver represents a proxy server.
- port (page 1149)
  Returns the receiver's port.
- protocol (page 1149)
  Returns the receiver's protocol.
- proxyType (page 1149)
  Returns the receiver's proxy type.
- realm (page 1149)
  Returns the receiver's authentication realm
- receivesCredentialSecurely (page 1150)
  Returns whether the credentials for the protection space can be sent securely.

Instance Methods

authenticationMethod

Returns the authentication method used by the receiver.

```
- (NSString *)authenticationMethod
```

Return Value

The authentication method used by the receiver. The supported authentication methods are listed in "Constants" (page 1150).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

host

Returns the receiver's host.

```
- (NSString *)host
```

Return Value
The receiver's host.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

initWithHost:port:protocol:realm:authenticationMethod:

Initializes a protection space object.

```
- (id)initWithHost:(NSString *)host port:(NSInteger)port protocol:(NSString
    *)protocol realm:(NSString *)realm authenticationMethod:(NSString
    *)authenticationMethod
```

Parameters

host
The host name for the protection space object.

port
The port for the protection space object. If *port* is 0 the default port for the specified protocol is used, for example, port 80 for HTTP. Note that servers can, and do, treat these values differently.

protocol
The protocol for the protection space object. The value of *protocol* is equivalent to the scheme for a URL in the protection space, for example, "http", "https", "ftp", etc.

realm
A string indicating a protocol specific subdivision of the host. *realm* may be `nil` if there is no specified realm or if the protocol doesn't support realms.

authenticationMethod
The type of authentication to use. *authenticationMethod* should be set to one of the values in "Constants" (page 1150) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithProxyHost:port:type:realm:authenticationMethod: (page 1148)

Declared In
NSURLProtectionSpace.h

initWithProxyHost:port:type:realm:authenticationMethod:

Initializes a protection space object representing a proxy server.

```
- (id)initWithProxyHost:(NSString *)host port:(NSInteger)port type:(NSString
    *)proxyType realm:(NSString *)realm authenticationMethod:(NSString
    *)authenticationMethod
```

Parameters

host
The host of the proxy server for the protection space object.

port
The port for the protection space object. If *port* is 0 the default port for the specified proxy type is used, for example, port 80 for HTTP. Note that servers can, and do, treat these values differently.

proxyType
The type of proxy server. The value of *proxyType* should be set to one of the values specified in "Constants" (page 1150).

realm
A string indicating a protocol specific subdivision of the host. *realm* may be `nil` if there is no specified realm or if the protocol doesn't support realms.

authenticationMethod
The type of authentication to use. *authenticationMethod* should be set to one of the values in "Constants" (page 1150) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `initWithHost:port:protocol:realm:authenticationMethod:` (page 1147)

Declared In
NSURLProtectionSpace.h

isProxy

Returns whether the receiver represents a proxy server.

```
- (BOOL)isProxy
```

Return Value
`YES` if the receiver represents a proxy server, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

port

Returns the receiver's port.

```
- (NSInteger)port
```

Return Value
The receiver's port.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

protocol

Returns the receiver's protocol.

```
- (NSString *)protocol
```

Return Value
The receiver's protocol, or `nil` if the receiver represents a proxy protection space.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

proxyType

Returns the receiver's proxy type.

```
- (NSString *)proxyType
```

Return Value
The receiver's proxy type, or `nil` if the receiver does not represent a proxy protection space. The supported proxy types are listed in "Constants" (page 1150).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

realm

Returns the receiver's authentication realm

```
- (NSString *)realm
```

Return Value
The receiver's authentication realm, or `nil` if no realm has been set.

Discussion
A realm is generally only specified for HTTP and HTTPS authentication.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

receivesCredentialSecurely

Returns whether the credentials for the protection space can be sent securely.

```
- (BOOL)receivesCredentialSecurely
```

Return Value
YES if the credentials for the protection space represented by the receiver can be sent securely, NO otherwise.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtectionSpace.h

Constants

NSURLProtectionSpace Proxy Types

These constants describe the supported proxy types used in initWithProxyHost:port:type:realm:authenticationMethod: (page 1148) and returned by proxyType (page 1149).

```
extern NSString *NSURLProtectionSpaceHTTPProxy;
extern NSString *NSURLProtectionSpaceHTTPSProxy;
extern NSString *NSURLProtectionSpaceFTPProxy;
extern NSString *NSURLProtectionSpaceSOCKSProxy;
```

Constants

NSURLProtectionSpaceHTTPProxy

The proxy type for HTTP proxies.

Available in iPhone OS 2.0 and later.

Declared in NSURLProtectionSpace.h

NSURLProtectionSpaceHTTPSProxy

The proxy type for HTTPS proxies.

Available in iPhone OS 2.0 and later.

Declared in NSURLProtectionSpace.h

`NSURLProtectionSpaceFTPProxy`
The proxy type for FTP proxies.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

`NSURLProtectionSpaceSOCKSProxy`
The proxy type for SOCKS proxies.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

NSURLProtectionSpace Authentication Methods

These constants describe the available authentication methods used in `initWithHost:port:protocol:realm:authenticationMethod:` (page 1147), `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1148) and returned by `authenticationMethod` (page 1146).

```
extern NSString *NSURLAuthenticationMethodDefault;
extern NSString *NSURLAuthenticationMethodHTTPBasic;
extern NSString *NSURLAuthenticationMethodHTTPDigest;
extern NSString *NSURLAuthenticationMethodHTMLForm;
```

Constants
`NSURLAuthenticationMethodDefault`
Use the default authentication method for a protocol.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

`NSURLAuthenticationMethodHTTPBasic`
Use HTTP basic authentication for this protection space.

This is equivalent to `NSURLAuthenticationMethodDefault` for HTTP.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

`NSURLAuthenticationMethodHTTPDigest`
Use HTTP digest authentication for this protection space.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

`NSURLAuthenticationMethodHTMLForm`
Use HTML form authentication for this protection space.

This authentication method can apply to any protocol.

Available in iPhone OS 2.0 and later.

Declared in `NSURLProtectionSpace.h`

Availability
Available in Mac OS X v10.2 with Safari 1.0 installed.
Available in Mac OS X v10.2.7 and later.

Declared In
`NSURLProtectionSpace.h`

NSURLProtocol Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLProtocol.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSURLProtocol` is an abstract class that provides the basic structure for performing protocol-specific loading of URL data. Concrete subclasses handle the specifics associated with one or more protocols or URL schemes.

An application should never need to directly instantiate an `NSURLProtocol` subclass. The instance of the appropriate `NSURLProtocol` subclass for an `NSURLRequest` is created by `NSURLConnection` when a download is started.

The `NSURLProtocolClient` protocol describes the methods an implementation uses to drive the URL loading system from a `NSURLProtocol` subclass.

To support customization of protocol-specific requests, protocol implementors are encouraged to provide categories on `NSURLRequest` and `NSMutableURLRequest`. Protocol implementors who need to extend the capabilities of `NSURLRequest` and `NSMutableURLRequest` in this way can store and retrieve protocol-specific request data by using `NSURLProtocol`'s class methods `propertyForKey:inRequest:` (page 1156) and `setProperty:forKey:inRequest:` (page 1158).

An essential responsibility for a protocol implementor is creating a `NSURLResponse` for each request it processes successfully. A protocol implementor may wish to create a custom, mutable `NSURLResponse` class to provide protocol specific information.

Tasks

Creating Protocol Objects

`- initWithRequest:cachedResponse:client:` (page 1160)
Initializes an `NSURLProtocol` object.

Registering and Unregistering Protocol Classes

`+ registerClass:` (page 1157)
Attempts to register a subclass of `NSURLProtocol`, making it visible to the URL loading system.

`+ unregisterClass:` (page 1159)
Unregisters the specified subclass of `NSURLProtocol`.

Getting and Setting Request Properties

`+ propertyForKey:inRequest:` (page 1156)
Returns the property associated with the specified key in the specified request.

`+ setProperty:forKey:inRequest:` (page 1158)
Sets the property associated with the specified key in the specified request.

`+ removePropertyForKey:inRequest:` (page 1157)
Removes the property associated with the specified key in the specified request.

Determining If a Subclass Can Handle a Request

`+ canInitWithRequest:` (page 1155)
Returns whether the protocol subclass can handle the specified request.

Providing a Canonical Version of a Request

`+ canonicalRequestForRequest:` (page 1156)
Returns a canonical version of the specified request.

Determining If Requests Are Cache Equivalent

`+ requestIsCacheEquivalent:toRequest:` (page 1158)
Returns whether two requests are equivalent for cache purposes.

Starting and Stopping Downloads

`- startLoading` (page 1161)
Starts protocol-specific loading of the request.

`- stopLoading` (page 1161)
Stops protocol-specific loading of the request.

Getting Protocol Attributes

`- cachedResponse` (page 1159)
Returns the receiver's cached response.

`- client` (page 1159)
Returns the object the receiver uses to communicate with the URL loading system.

`- request` (page 1160)
Returns the receiver's request.

Class Methods

canInitWithRequest:

Returns whether the protocol subclass can handle the specified request.

```
+ (BOOL)canInitWithRequest:(NSURLRequest *)request
```

Parameters

request
The request to be handled.

Return Value

`YES` if the protocol subclass can handle *request*, otherwise `NO`.

Discussion

A subclass should inspect *request* and determine whether or not the implementation can perform a load with that request.

This is an abstract method and subclasses must provide an implementation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLProtocol.h`

canonicalRequestForRequest:

Returns a canonical version of the specified request.

```
+ (NSURLRequest *)canonicalRequestForRequest:(NSURLRequest *)request
```

Parameters

request

The request whose canonical version is desired.

Return Value

The canonical form of *request*.

Discussion

It is up to each concrete protocol implementation to define what "canonical" means. A protocol should guarantee that the same input request always yields the same canonical form.

Special consideration should be given when implementing this method, because the canonical form of a request is used to lookup objects in the URL cache, a process which performs equality checks between `NSURLRequest` objects.

This is an abstract method and subclasses must provide an implementation.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLProtocol.h`

propertyForKey:inRequest:

Returns the property associated with the specified key in the specified request.

```
+ (id)propertyForKey:(NSString *)key inRequest:(NSURLRequest *)request
```

Parameters

key

The key of the desired property.

request

The request whose properties are to be queried.

Return Value

The property associated with *key*, or `nil` if no property has been stored for *key*.

Discussion

This method provides an interface for protocol implementors to access protocol-specific information associated with `NSURLRequest` objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ setProperty:forKey:inRequest:` (page 1158)

Declared In

`NSURLProtocol.h`

registerClass:

Attempts to register a subclass of `NSURLProtocol`, making it visible to the URL loading system.

```
+ (BOOL)registerClass:(Class)protocolClass
```

Parameters

protocolClass
The subclass of NSURLProtocol to register.

Return Value

`YES` if the registration is successful, `NO` otherwise. The only failure condition is if *protocolClass* is not a subclass of NSURLProtocol.

Discussion

When the URL loading system begins to load a request, each registered protocol class is consulted in turn to see if it can be initialized with the specified request. The first `NSURLProtocol` subclass to return `YES` when sent a `canInitWithRequest:` (page 1155) message is used to perform the URL load. There is no guarantee that all registered protocol classes will be consulted.

Classes are consulted in the reverse order of their registration. A similar design governs the process to create the canonical form of a request with `canonicalRequestForRequest:` (page 1156).

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ unregisterClass:` (page 1159)

Declared In

`NSURLProtocol.h`

removePropertyForKey:inRequest:

Removes the property associated with the specified key in the specified request.

```
+ (void)removePropertyForKey:((NSString *)key inRequest:(NSMutableURLRequest
    *)request
```

Parameters

key
The key whose value should be removed.

request
The request from which to remove the property value.

Discussion

This method is used to provide an interface for protocol implementors to customize protocol-specific information associated with `NSMutableURLRequest` objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

`+ propertyForKey:inRequest:` (page 1156)

Declared In
NSURLProtocol.h

requestIsCacheEquivalent:toRequest:

Returns whether two requests are equivalent for cache purposes.

```
+ (BOOL)requestIsCacheEquivalent:(NSURLRequest *)aRequest toRequest:(NSURLRequest
    *)bRequest
```

Parameters

aRequest
The request to compare with *bRequest*.

bRequest
The request to compare with *aRequest*.

Return Value
YES if *aRequest* and *bRequest* are equivalent for cache purposes, NO otherwise. Requests are considered equivalent for cache purposes if and only if they would be handled by the same protocol and that protocol declares them equivalent after performing implementation-specific checks.

Discussion
The NSURLProtocol implementation of this method compares the URLs of the requests to determine if the requests should be considered equivalent. Subclasses can override this method to provide protocol-specific comparisons.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

setProperty:forKey:inRequest:

Sets the property associated with the specified key in the specified request.

```
+ (void)setProperty:(id)value forKey:(NSString *)key inRequest:(NSMutableURLRequest
    *)request
```

Parameters

value
The value to set for the specified property.

key
The key for the specified property.

request
The request for which to create the property.

Discussion
This method is used to provide an interface for protocol implementors to customize protocol-specific information associated with NSMutableURLRequest objects.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ propertyForKey:inRequest: (page 1156)

Declared In
NSURLProtocol.h

unregisterClass:

Unregisters the specified subclass of NSURLProtocol.

```
+ (void)unregisterClass:(Class)protocolClass
```

Parameters

protocolClass
The subclass of NSURLProtocol to unregister.

Discussion
After this method is invoked, *protocolClass* is no longer consulted by the URL loading system.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ registerClass: (page 1157)

Declared In
NSURLProtocol.h

Instance Methods

cachedResponse

Returns the receiver's cached response.

```
- (NSCachedURLResponse *)cachedResponse
```

Return Value
The receiver's cached response.

Discussion
Subclasses must implement this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

client

Returns the object the receiver uses to communicate with the URL loading system.

```
- (id < NSURLProtocolClient >)client
```

Return Value
The object the receiver uses to communicate with the URL loading system.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

initWithRequest:cachedResponse:client:

Initializes an NSURLProtocol object.

```
- (id)initWithRequest:(NSURLRequest *)request cachedResponse:(NSCachedURLResponse
    *)cachedResponse client:(id < NSURLProtocolClient >)client
```

Parameters
request
: The URL request for the URL protocol object.

cachedResponse
: A cached response for the request; may be nil if there is no existing cached response for the request.

client
: An object that provides an implementation of the NSURLProtocolClient protocol that the receiver uses to communicate with the URL loading system.

Discussion
Subclasses should override this method to do any custom initialization. An application should never explicitly call this method.

This is the designated intializer for NSURLProtocol.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

request

Returns the receiver’s request.

```
- (NSURLRequest *)request
```

Return Value
The receiver's request.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

startLoading

Starts protocol-specific loading of the request.

```
- (void)startLoading
```

Discussion
When this method is called, the subclass implementation should start loading the request, providing feedback to the URL loading system via the `NSURLProtocolClient` protocol.

Subclasses must implement this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `stopLoading` (page 1161)

Declared In
NSURLProtocol.h

stopLoading

Stops protocol-specific loading of the request.

```
- (void)stopLoading
```

Discussion
When this method is called, the subclass implementation should stop loading a request. This could be in response to a cancel operation, so protocol implementations must be able to handle this call while a load is in progress.

Subclasses must implement this method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `startLoading` (page 1161)

Declared In
NSURLProtocol.h

NSURLRequest Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLRequest.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSURLRequest` objects represent a URL load request in a manner independent of protocol and URL scheme.

NSURLRequest encapsulates two basic data elements of a load request: the URL to load, and the policy to use when consulting the URL content cache made available by the implementation.

NSURLRequest is designed to be extended to support additional protocols by adding categories that access protocol specific values from a property object using NSURLProtocol's `propertyForKey:inRequest:` (page 1156) and `setProperty:forKey:inRequest:` (page 1158) methods.

The mutable subclass of NSURLRequest is NSMutableURLRequest.

Adopted Protocols

NSCopying
- copyWithZone: (page 1250)

NSMutableCopying
- mutableCopyWithZone: (page 1300)

Tasks

Creating Requests

+ requestWithURL: (page 1165)
Creates and returns a URL request for a specified URL with default cache policy and timeout value.

- initWithURL: (page 1168)
Returns a URL request for a specified URL with default cache policy and timeout value.

+ requestWithURL:cachePolicy:timeoutInterval: (page 1165)
Creates and returns an initialized URL request with specified values.

- initWithURL:cachePolicy:timeoutInterval: (page 1168)
Returns an initialized URL request with specified values.

Getting Request Properties

- cachePolicy (page 1166)
Returns the receiver's cache policy.

- mainDocumentURL (page 1169)
Returns the main document URL associated with the request.

- timeoutInterval (page 1169)
Returns the receiver's timeout interval, in seconds.

- URL (page 1170)
Returns the request's URL.

Getting HTTP Request Properties

- allHTTPHeaderFields (page 1166)
Returns a dictionary containing all the receiver's HTTP header fields.

- HTTPBody (page 1167)
Returns the receiver's HTTP body data.

- HTTPBodyStream (page 1167)
  Returns the receiver's HTTP body stream.
- HTTPMethod (page 1167)
  Returns the receiver's HTTP request method.
- HTTPShouldHandleCookies (page 1168)
  Returns whether the default cookie handling will be used for this request.
- valueForHTTPHeaderField: (page 1170)
  Returns the value of the specified HTTP header field.

Class Methods

requestWithURL:

Creates and returns a URL request for a specified URL with default cache policy and timeout value.

```
+ (id)requestWithURL:(NSURL *)theURL
```

Parameters

theURL
The URL for the new request.

Return Value

The newly created URL request.

Discussion

The default cache policy is NSURLRequestUseProtocolCachePolicy and the default timeout interval is 60 seconds.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ requestWithURL:cachePolicy:timeoutInterval: (page 1165)

Declared In

NSURLRequest.h

requestWithURL:cachePolicy:timeoutInterval:

Creates and returns an initialized URL request with specified values.

```
+ (id)requestWithURL:(NSURL *)theURL cachePolicy:(NSURLRequestCachePolicy)cachePolicy
    timeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

theURL
The URL for the new request.

cachePolicy
The cache policy for the new request.

timeoutInterval
The timeout interval for the new request, in seconds.

Return Value
The newly created URL request.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithURL:cachePolicy:timeoutInterval: (page 1168)

Declared In
NSURLRequest.h

Instance Methods

allHTTPHeaderFields

Returns a dictionary containing all the receiver's HTTP header fields.

```
- (NSDictionary *)allHTTPHeaderFields
```

Return Value
A dictionary containing all the receiver's HTTP header fields.

Availability
Available in iPhone OS 2.0 and later.

See Also
- valueForHTTPHeaderField: (page 1170)

Declared In
NSURLRequest.h

cachePolicy

Returns the receiver's cache policy.

```
- (NSURLRequestCachePolicy)cachePolicy
```

Return Value
The receiver's cache policy.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

HTTPBody

Returns the receiver's HTTP body data.

```
- (NSData *)HTTPBody
```

Return Value
The receiver's HTTP body data.

Discussion
This data is sent as the message body of a request, as in an HTTP `POST` request.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLRequest.h`

HTTPBodyStream

Returns the receiver's HTTP body stream.

```
- (NSInputStream *)HTTPBodyStream
```

Return Value
The receiver's HTTP body stream, or `nil` if it has not been set. The returned stream is for examination only, it is not safe to manipulate the stream in any way.

Discussion
The receiver will have either an HTTP body or an HTTP body stream, only one may be set for a request. A HTTP body stream is preserved when copying an NSURLRequest object, but is lost when a request is archived using the NSCoding protocol.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLRequest.h`

HTTPMethod

Returns the receiver's HTTP request method.

```
- (NSString *)HTTPMethod
```

Return Value
The receiver's HTTP request method.

Discussion
The default HTTP method is "GET".

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

HTTPShouldHandleCookies

Returns whether the default cookie handling will be used for this request.

```
- (BOOL)HTTPShouldHandleCookies
```

Return Value
YES if the default cookie handling will be used for this request, NO otherwise.

Discussion
The default is YES.

Special Considerations
In Mac OS X v10.2 with Safari 1.0 the value set by this method is not respected by the framework.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

initWithURL:

Returns a URL request for a specified URL with default cache policy and timeout value.

```
- (id)initWithURL:(NSURL *)theURL
```

Parameters
theURL
The URL for the request.

Return Value
The initialized URL request.

Discussion
The default cache policy is NSURLRequestUseProtocolCachePolicy and the default timeout interval is 60 seconds.

Availability
Available in iPhone OS 2.0 and later.

See Also
- initWithURL:cachePolicy:timeoutInterval: (page 1168)

Declared In
NSURLRequest.h

initWithURL:cachePolicy:timeoutInterval:

Returns an initialized URL request with specified values.

```
- (id)initWithURL:(NSURL *)theURL cachePolicy:(NSURLRequestCachePolicy)cachePolicy
    timeoutInterval:(NSTimeInterval)timeoutInterval
```

Parameters

theURL

The URL for the request.

cachePolicy

The cache policy for the request.

timeoutInterval

The timeout interval for the request, in seconds.

Return Value

The initialized URL request.

Discussion

This is the designated initializer for `NSURLRequest`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- initWithURL:` (page 1168)

Declared In

`NSURLRequest.h`

mainDocumentURL

Returns the main document URL associated with the request.

```
- (NSURL *)mainDocumentURL
```

Return Value

The main document URL associated with the request.

Discussion

This URL is used for the cookie “same domain as main document” policy.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLRequest.h`

timeoutInterval

Returns the receiver’s timeout interval, in seconds.

```
- (NSTimeInterval)timeoutInterval
```

Return Value

The receiver's timeout interval, in seconds.

Discussion
If during a connection attempt the request remains idle for longer than the timeout interval, the request is considered to have timed out.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

URL

Returns the request's URL.

```
- (NSURL *)URL
```

Return Value
The request's URL.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

valueForHTTPHeaderField:

Returns the value of the specified HTTP header field.

```
- (NSString *)valueForHTTPHeaderField:(NSString *)field
```

Parameters

field

The name of the header field whose value is to be returned. In keeping with the HTTP RFC, HTTP header field names are case-insensitive.

Return Value
The value associated with the header field *field*, or nil if there is no corresponding header field.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLRequest.h

Constants

NSURLRequestCachePolicy

These constants are used to specify interaction with the cached responses.

```
enum
{
    NSURLRequestUseProtocolCachePolicy = 0,
    NSURLRequestReloadIgnoringLocalCacheData = 1,
    NSURLRequestReloadIgnoringLocalAndRemoteCacheData =4,
   NSURLRequestReloadIgnoringCacheData = NSURLRequestReloadIgnoringLocalCacheData,
    NSURLRequestReturnCacheDataElseLoad = 2,
    NSURLRequestReturnCacheDataDontLoad = 3,
    NSURLRequestReloadRevalidatingCacheData = 5
};
typedef NSUInteger NSURLRequestCachePolicy;
```

Constants

`NSURLRequestUseProtocolCachePolicy`

Specifies that the caching logic defined in the protocol implementation, if any, is used for a particular URL load request. This is the default policy for URL load requests.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReloadIgnoringLocalCacheData`

Specifies that the data for the URL load should be loaded from the originating source. No existing cache data should be used to satisfy a URL load request.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReloadIgnoringLocalAndRemoteCacheData`

Specifies that not only should the local cache data be ignored, but that proxies and other intermediates should be instructed to disregard their caches so far as the protocol allows.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReloadIgnoringCacheData`

Replaced by `NSURLRequestReloadIgnoringLocalCacheData` (page 1171).

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReturnCacheDataElseLoad`

Specifies that the existing cached data should be used to satisfy the request, regardless of its age or expiration date. If there is no existing data in the cache corresponding the request, the data is loaded from the originating source.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReturnCacheDataDontLoad`

Specifies that the existing cache data should be used to satisfy a request, regardless of its age or expiration date. If there is no existing data in the cache corresponding to a URL load request, no attempt is made to load the data from the originating source, and the load is considered to have failed. This constant specifies a behavior that is similar to an “offline” mode.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

`NSURLRequestReloadRevalidatingCacheData`

Specifies that the existing cache data may be used provided the origin source confirms its validity, otherwise the URL is loaded from the origin source.

Available in iPhone OS 2.0 and later.

Declared in `NSURLRequest.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSURLRequest.h`

NSURLResponse Class Reference

Inherits from:	NSObject
Conforms to:	NSCopying NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLResponse.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

`NSURLResponse` declares the programmatic interface for an object that accesses the response returned by an `NSURLRequest` instance.

`NSURLResponse` encapsulates the metadata associated with a URL load in a manner independent of protocol and URL scheme.

`NSHTTPURLResponse` is a subclass of `NSURLResponse` that provides methods for accessing information specific to HTTP protocol responses. An `NSHTTPURLResponse` object represents a response to an HTTP URL load request.

Note: `NSURLResponse` objects do not contain the actual bytes representing the content of a URL. See `NSURLConnection` for more information about receiving the content data for a URL load.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1246)
- `initWithCoder:` (page 1246)

NSCopying
- `copyWithZone:` (page 1250)

Tasks

Creating a Response

- `initWithURL:MIMEType:expectedContentLength:textEncodingName:` (page 1175)
  Returns an initialized `NSURLResponse` object with the URL, MIME type, length, and text encoding set to given values.

Getting the Response Properties

- `expectedContentLength` (page 1174)
  Returns the receiver's expected content length
- `suggestedFilename` (page 1176)
  Returns a suggested filename for the response data.
- `MIMEType` (page 1175)
  Returns the receiver's MIME type.
- `textEncodingName` (page 1176)
  Returns the name of the receiver's text encoding provided by the response's originating source.
- `URL` (page 1177)
  Returns the receiver's URL.

Instance Methods

expectedContentLength

Returns the receiver's expected content length

```
- (long long)expectedContentLength
```

Return Value
The receiver's expected content length, or `NSURLResponseUnknownLength` if the length can't be determined.

Discussion
Some protocol implementations report the content length as part of the response, but not all protocols guarantee to deliver that amount of data. Clients should be prepared to deal with more or less data.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

initWithURL:MIMEType:expectedContentLength:textEncodingName:

Returns an initialized `NSURLResponse` object with the URL, MIME type, length, and text encoding set to given values.

```
- (id)initWithURL:(NSURL *)URL MIMEType:(NSString *)MIMEType
    expectedContentLength:(NSInteger)length textEncodingName:(NSString *)name
```

Parameters
URL
: The URL for the new object.

MIMEType
: The MIME type.

length
: The expected content length.This value should be `-1` if the expected length is undetermined

name
: The text encoding name. This value may be `nil`.

Return Value
An initialized `NSURLResponse` object with the URL set to *URL*, the MIME type set to *MIMEType*, length set to *length*, and text encoding name set to *name*.

Discussion
This is the designated initializer for `NSURLResponse`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

MIMEType

Returns the receiver's MIME type.

```
- (NSString *)MIMEType
```

Return Value
The receiver's MIME type.

Discussion
The MIME type is often provided by the response's originating source. However, that value may be changed or corrected by a protocol implementation if it can be determined that the response's source reported the information incorrectly.

If the response's originating source does not provide a MIME type, an attempt to guess the MIME type may be made.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

suggestedFilename

Returns a suggested filename for the response data.

```
- (NSString *)suggestedFilename
```

Return Value
A suggested filename for the response data.

Discussion
The method tries to create a filename using the following, in order:

1. A filename specified using the content disposition header.
2. The last path component of the URL.
3. The host of the URL.

If the host of URL can't be converted to a valid filename, the filename "unknown" is used.

In most cases, this method appends the proper file extension based on the MIME type. This method will always return a valid filename regardless of whether or not the resource is saved to disk.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

textEncodingName

Returns the name of the receiver's text encoding provided by the response's originating source.

```
- (NSString *)textEncodingName
```

Return Value
The name of the receiver's text encoding provided by the response's originating source, or `nil` if no text encoding was provided by the protocol

Discussion
Clients can convert this string to an `NSStringEncoding` or a `CFStringEncoding` using the methods and functions available in the appropriate framework.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

URL
Returns the receiver's URL.

```
- (NSURL *)URL
```

Return Value
The receiver's URL.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSURLResponse.h`

Constants

Response Length Unknown Error
The following error code is returned by `expectedContentLength` (page 1174).

```
#define NSURLResponseUnknownLength ((long long)-1)
```

Constants
`NSURLResponseUnknownLength`

Returned when the response length cannot be determined in advance of receiving the data from the server. For example, `NSURLResponseUnknownLength` is returned when the server HTTP response does not include a Content-Length header.

Available in iPhone OS 2.0 and later.

Declared in `NSURLResponse.h`

NSUserDefaults Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSUserDefaults.h
Companion guide:	User Defaults Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSUserDefaults` class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences. For example, you can allow users to determine what units of measurement your application displays or how often documents are automatically saved. Applications record such preferences by assigning values to a set of parameters in a user's defaults database. The parameters are referred to as defaults since they're commonly used to determine an application's default state at startup or the way it acts by default.

At runtime, you use an `NSUserDefaults` object to read the defaults that your application uses from a user's defaults database. `NSUserDefaults` caches the information to avoid having to open the user's defaults database each time you need a default value. The `synchronize` (page 1198) method, which is automatically invoked at periodic intervals, keeps the in-memory cache in sync with a user's defaults database.

A default's value must be a property list, that is, an instance of (or for collections a combination of instances of): `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`. If you want to store any other type of object, you should typically archive it to create an instance of `NSData`. For more details, see *User Defaults Programming Topics for Cocoa*.

Values returned from `NSUserDefaults` are *immutable*, even if you set a mutable object as the value. For example, if you set a mutable string as the value for `"MyStringDefault"`, the string you later retrieve using `stringForKey:` (page 1197) will be immutable.

A defaults database is created automatically for each user. The `NSUserDefaults` class does not currently support per-host preferences. To do this, you must use the CFPreferences API (see *Preferences Utilities Reference*). However, `NSUserDefaults` correctly reads per-host preferences, so you can safely mix CFPreferences code with `NSUserDefaults` code.

If your application supports managed environments, you can use an `NSUserDefaults` object to determine which preferences are managed by an administrator for the benefit of the user. Managed environments correspond to computer labs or classrooms where an administrator or teacher may want to configure the systems in a particular way. In these situations, the teacher can establish a set of default preferences and force those preferences on users. If a preference is managed in this manner, applications should prevent users from editing that preference by disabling any appropriate controls.

The `NSUserDefaults` class is thread-safe.

Tasks

Getting the Shared NSUserDefaults Instance

`+ standardUserDefaults` (page 1183)
Returns the shared defaults object.

`+ resetStandardUserDefaults` (page 1183)
Synchronizes any changes made to the shared user defaults object and releases it from memory.

Initializing an NSUserDefaults Object

`- init` (page 1188)
Returns an `NSUserDefaults` object initialized with the defaults for the current user account.

`- initWithUser:` (page 1188)
Returns an `NSUserDefaults` object initialized with the defaults for the specified user account.

Getting a Default Value

- `arrayForKey:` (page 1184)
  Returns the array associated with the specified key.
- `boolForKey:` (page 1185)
  Returns the Boolean value associated with the specified key.
- `dataForKey:` (page 1185)
  Returns the data object associated with the specified key.
- `dictionaryForKey:` (page 1186)
  Returns the dictionary object associated with the specified key.
- `floatForKey:` (page 1187)
  Returns the floating-point value associated with the specified key.
- `integerForKey:` (page 1189)
  Returns the integer value associated with the specified key..
- `objectForKey:` (page 1189)
  Returns the object associated with the first occurrence of the specified default.
- `stringArrayForKey:` (page 1197)
  Returns the array of strings associated with the specified key.
- `stringForKey:` (page 1197)
  Returns the string associated with the specified key.

Setting and Removing Defaults

- `removeObjectForKey:` (page 1192)
  Removes the value of the specified default key in the standard application domain.
- `setBool:forKey:` (page 1194)
  Sets the value of the specified default key to a string containing a Boolean value.
- `setFloat:forKey:` (page 1194)
  Sets the value of the specified default key to a string containing a floating-point value.
- `setInteger:forKey:` (page 1195)
  Sets the value of the specified default key to a string containing an integer value.
- `setObject:forKey:` (page 1195)
  Sets the value of the specified default key in the standard application domain.

Registering Defaults

- `registerDefaults:` (page 1192)
  Adds the contents the specified dictionary to the registration domain.

Maintaining Persistent Domains

- `synchronize` (page 1198)
  Writes any modifications to the persistent domains to disk and updates all unmodified persistent domains to what is on disk.

- `persistentDomainForName:` (page 1191)
  Returns a dictionary containing the keys and values in the specified persistent domain.
- `persistentDomainNames` (page 1191)
  Returns an array of the current persistent domain names.
- `removePersistentDomainForName:` (page 1193)
  Removes the contents of the specified persistent domain from the user's defaults.
- `setPersistentDomain:forName:` (page 1196)
  Sets the dictionary for the specified persistent domain.

Accessing Managed Environment Keys

- `objectIsForcedForKey:` (page 1190)
  Returns a Boolean value indicating whether the specified key is managed by an administrator.
- `objectIsForcedForKey:inDomain:` (page 1190)
  Returns a Boolean value indicating whether the key in the specified domain is managed by an administrator.

Managing the Search List

- `dictionaryRepresentation` (page 1187)
  Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

Maintaining Volatile Domains

- `removeVolatileDomainForName:` (page 1193)
  Removes the specified volatile domain from the user's defaults.
- `setVolatileDomain:forName:` (page 1196)
  Sets the dictionary for the specified volatile domain.
- `volatileDomainForName:` (page 1199)
  Returns the dictionary for the specified volatile domain.
- `volatileDomainNames` (page 1199)
  Returns an array of the current volatile domain names.

Maintaining Suites

- `addSuiteNamed:` (page 1184)
  Inserts the specified domain name into the receiver's search list.
- `removeSuiteNamed:` (page 1193)
  Removes the specified domain name from the receiver's search list.

Class Methods

resetStandardUserDefaults

Synchronizes any changes made to the shared user defaults object and releases it from memory.

```
+ (void)resetStandardUserDefaults
```

Discussion

A subsequent invocation of `standardUserDefaults` (page 1183) creates a new shared user defaults object with the standard search list.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSUserDefaults.h`

standardUserDefaults

Returns the shared defaults object.

```
+ (NSUserDefaults *)standardUserDefaults
```

Return Value

The shared defaults object.

Discussion

If the shared defaults object does not exist yet, it is created with a search list containing the names of the following domains, in this order:

- `NSArgumentDomain`, consisting of defaults parsed from the application's arguments
- A domain identified by the application's bundle identifier
- `NSGlobalDomain`, consisting of defaults meant to be seen by all applications
- Separate domains for each of the user's preferred languages
- `NSRegistrationDomain`, a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful

The defaults are initialized for the current user. Subsequent modifications to the standard search list remain in effect even when this method is invoked again—the search list is guaranteed to be standard only the first time this method is invoked. The shared instance is provided as a convenience—you can create custom instances using `alloc` along with `initWithUser:` (page 1188) or `init` (page 1188).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSUserDefaults.h`

Instance Methods

addSuiteNamed:

Inserts the specified domain name into the receiver's search list.

```
- (void)addSuiteNamed:(NSString *)suiteName
```

Parameters

suiteName

The domain name to insert. This domain is inserted after the application domain.

Discussion

The *suiteName* domain is similar to a bundle identifier string, but is not tied to a particular application or bundle. A suite can be used to hold preferences that are shared between multiple applications.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ standardUserDefaults (page 1183)
- removeSuiteNamed: (page 1193)

Declared In

NSUserDefaults.h

arrayForKey:

Returns the array associated with the specified key.

```
- (NSArray *)arrayForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The array associated with the specified key, or `nil` if the key does not exist or its value is not an `NSArray` object.

Special Considerations

The returned array and its contents are immutable, even if the values you originally set were mutable.

Availability

Available in iPhone OS 2.0 and later.

See Also

- boolForKey: (page 1185)
- dataForKey: (page 1185)
- dictionaryForKey: (page 1186)
- floatForKey: (page 1187)
- integerForKey: (page 1189)

- objectForKey: (page 1189)
- stringArrayForKey: (page 1197)
- stringForKey: (page 1197)

Declared In
NSUserDefaults.h

boolForKey:

Returns the Boolean value associated with the specified key.

```
- (BOOL)boolForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
YES if the value associated with *defaultName* is an NSString containing the word "yes" in uppercase or lowercase or responds to the intValue message by returning a nonzero value; otherwise NO.

Availability
Available in iPhone OS 2.0 and later.

See Also
- arrayForKey: (page 1184)
- dataForKey: (page 1185)
- dictionaryForKey: (page 1186)
- floatForKey: (page 1187)
- integerForKey: (page 1189)
- objectForKey: (page 1189)
- stringArrayForKey: (page 1197)
- stringForKey: (page 1197)

Declared In
NSUserDefaults.h

dataForKey:

Returns the data object associated with the specified key.

```
- (NSData *)dataForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The data object associated with the specified key, or nil if the key does not exist or its value is not an NSData object.

Special Considerations

The returned data object is immutable, even if the value you originally set was a mutable data object.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `floatForKey:` (page 1187)
- `integerForKey:` (page 1189)
- `objectForKey:` (page 1189)
- `stringArrayForKey:` (page 1197)
- `stringForKey:` (page 1197)

Declared In

`NSUserDefaults.h`

dictionaryForKey:

Returns the dictionary object associated with the specified key.

```
- (NSDictionary *)dictionaryForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The dictionary object associated with the specified key, or `nil` if the key does not exist or its value is not an `NSDictionary` object.

Special Considerations

The returned dictionary and its contents are immutable, even if the values you originally set were mutable.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `floatForKey:` (page 1187)
- `integerForKey:` (page 1189)
- `objectForKey:` (page 1189)
- `stringArrayForKey:` (page 1197)
- `stringForKey:` (page 1197)

Declared In
NSUserDefaults.h

dictionaryRepresentation

Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

```
- (NSDictionary *)dictionaryRepresentation
```

Return Value
A dictionary containing the keys. The keys are names of defaults and the value corresponding to each key is a property list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`).

Discussion
As with `objectForKey:` (page 1189), key-value pairs in domains that are earlier in the search list take precedence. The combined result does not preserve information about which domain each entry came from.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSUserDefaults.h

floatForKey:

Returns the floating-point value associated with the specified key.

```
- (float)floatForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The floating-point value associated with the specified key. If the string storing the value does not exist, this method returns `0`; otherwise, the string object is sent a `floatValue` message and the resulting value is returned.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `integerForKey:` (page 1189)
- `objectForKey:` (page 1189)
- `stringArrayForKey:` (page 1197)
- `stringForKey:` (page 1197)

Declared In
NSUserDefaults.h

init

Returns an NSUserDefaults object initialized with the defaults for the current user account.

```
- (id)init
```

Return Value
An initialized NSUserDefaults object whose argument and registration domains are already set up.

Discussion
This method does not put anything in the search list. Invoke it only if you've allocated your own NSUserDefaults instance instead of using the shared one.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ standardUserDefaults (page 1183)

Declared In
NSUserDefaults.h

initWithUser:

Returns an NSUserDefaults object initialized with the defaults for the specified user account.

```
- (id)initWithUser:(NSString *)username
```

Parameters
username
The name of the user account.

Return Value
An initialized NSUserDefaults object whose argument and registration domains are already set up. If the current user does not have access to the specified user account, this method returns nil.

Discussion
This method does not put anything in the search list. Invoke it only if you've allocated your own NSUserDefaults instance instead of using the shared one.

You do not normally use this method to initialize an instance of NSUserDefaults. Applications used by a superuser might use this method to update the defaults databases for a number of users. The user who started the application must have appropriate access (read, write, or both) to the defaults database of the new user, or this method returns nil.

Availability
Available in iPhone OS 2.0 and later.

See Also
+ standardUserDefaults (page 1183)

Declared In
NSUserDefaults.h

integerForKey:

Returns the integer value associated with the specified key..

```
- (NSInteger)integerForKey:(NSString *)defaultName
```

Parameters

defaultName
A key in the current user's defaults database.

Return Value
The integer value associated with the specified key. If the string storing the value does not exist, this method returns `0`; otherwise, the string object is sent an `intValue` message and the resulting value is returned.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `floatForKey:` (page 1187)
- `objectForKey:` (page 1189)
- `stringArrayForKey:` (page 1197)
- `stringForKey:` (page 1197)

Declared In
NSUserDefaults.h

objectForKey:

Returns the object associated with the first occurrence of the specified default.

```
- (id)objectForKey:(NSString *)defaultName
```

Parameters

defaultName
A key in the current user's defaults database.

Return Value
The object associated with the specified key, or `nil` if the key was not found.

Discussion
This method searches the domains included in the search list in the order they are listed.

Special Considerations

The returned object is immutable, even if the value you originally set was mutable.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `floatForKey:` (page 1187)
- `stringArrayForKey:` (page 1197)
- `stringForKey:` (page 1197)

Declared In
`NSUserDefaults.h`

objectIsForcedForKey:

Returns a Boolean value indicating whether the specified key is managed by an administrator.

```
- (BOOL)objectIsForcedForKey:(NSString *)key
```

Parameters

key
The key whose status you want to check.

Return Value
`YES` if the value of the specified key is managed by an administrator, otherwise `NO`.

Discussion
This method assumes that the key is a preference associated with the current user and application. For managed keys, the application should disable any user interface that allows the user to modify the value of *key*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `objectIsForcedForKey:inDomain:` (page 1190)

Declared In
`NSUserDefaults.h`

objectIsForcedForKey:inDomain:

Returns a Boolean value indicating whether the key in the specified domain is managed by an administrator.

```
- (BOOL)objectIsForcedForKey:(NSString *)key inDomain:(NSString *)domain
```

Parameters

key
The key whose status you want to check.

domain
The domain of the key.

Return Value
YES if the key is managed by an administrator in the specified domain, otherwise NO.

Discussion
This method assumes that the key is a preference associated with the current user. For managed keys, the application should disable any user interface that allows the user to modify the value of *key*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- objectIsForcedForKey: (page 1190)

Declared In
NSUserDefaults.h

persistentDomainForName:

Returns a dictionary containing the keys and values in the specified persistent domain.

```
- (NSDictionary *)persistentDomainForName:(NSString *)domainName
```

Parameters
domainName
The domain whose keys and values you want. This value should be equal to your application's bundle identifier.

Return Value
A dictionary containing the keys. The keys are names of defaults and the value corresponding to each key is a property list object (NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary).

Availability
Available in iPhone OS 2.0 and later.

See Also
- removePersistentDomainForName: (page 1193)
- setPersistentDomain:forName: (page 1196)

Declared In
NSUserDefaults.h

persistentDomainNames

Returns an array of the current persistent domain names.

```
- (NSArray *)persistentDomainNames
```

Return Value
An array of NSString objects containing the domain names.

Discussion
You can get the keys and values for each domain by passing the returned domain names to the `persistentDomainForName:` (page 1191) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removePersistentDomainForName:` (page 1193)
- `setPersistentDomain:forName:` (page 1196)

Declared In
`NSUserDefaults.h`

registerDefaults:
Adds the contents the specified dictionary to the registration domain.

```
- (void)registerDefaults:(NSDictionary *)dictionary
```

Parameters
dictionary
The dictionary of keys and values you want to register.

Discussion
If there is no registration domain, one is created using the specified dictionary, and `NSRegistrationDomain` is added to the end of the search list.

The contents of the registration domain are not written to disk; you need to call this method each time your application starts. You can place a plist file in the application's Resources directory and call `registerDefaults:` with the contents that you read in from that file.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSUserDefaults.h`

removeObjectForKey:
Removes the value of the specified default key in the standard application domain.

```
- (void)removeObjectForKey:(NSString *)defaultName
```

Parameters
defaultName
The key whose value you want to remove.

Discussion
Removing a default has no effect on the value returned by the `objectForKey:` (page 1189) method if the same key exists in a domain that precedes the standard application domain in the search list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setObject:forKey: (page 1195)

Declared In
NSUserDefaults.h

removePersistentDomainForName:

Removes the contents of the specified persistent domain from the user's defaults.

```
- (void)removePersistentDomainForName:(NSString *)domainName
```

Parameters

domainName
The domain whose keys and values you want. This value should be equal to your application's bundle identifier.

Discussion
When a persistent domain is changed, an NSUserDefaultsDidChangeNotification (page 1200) is posted.

Availability
Available in iPhone OS 2.0 and later.

See Also
- setPersistentDomain:forName: (page 1196)

Declared In
NSUserDefaults.h

removeSuiteNamed:

Removes the specified domain name from the receiver's search list.

```
- (void)removeSuiteNamed:(NSString *)suiteName
```

Parameters

suiteName
The domain name to remove.

Availability
Available in iPhone OS 2.0 and later.

See Also
- addSuiteNamed: (page 1184)

Declared In
NSUserDefaults.h

removeVolatileDomainForName:

Removes the specified volatile domain from the user's defaults.

```
- (void)removeVolatileDomainForName:(NSString *)domainName
```

Parameters

domainName

The volatile domain you want to remove.

Availability

Available in iPhone OS 2.0 and later.

See Also

- setVolatileDomain:forName: (page 1196)

Declared In

NSUserDefaults.h

setBool:forKey:

Sets the value of the specified default key to a string containing a Boolean value.

```
- (void)setBool:(BOOL)value forKey:(NSString *)defaultName
```

Parameters

value

The Boolean value to store in the defaults database. This value is converted to an NSString object before being stored in the defaults database.

defaultName

The key with which to associate with the value.

Discussion

Invokes setObject:forKey: (page 1195) as part of its implementation.

Availability

Available in iPhone OS 2.0 and later.

See Also

- boolForKey: (page 1185)

Declared In

NSUserDefaults.h

setFloat:forKey:

Sets the value of the specified default key to a string containing a floating-point value.

```
- (void)setFloat:(float)value forKey:(NSString *)defaultName
```

Parameters

value

The floating-point value to store in the defaults database. This value is converted to an NSString object before being stored in the defaults database.

defaultName

The key with which to associate with the value.

Discussion
Invokes `setObject:forKey:` (page 1195) as part of its implementation.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- floatForKey:` (page 1187)

Declared In
`NSUserDefaults.h`

setInteger:forKey:

Sets the value of the specified default key to a string containing an integer value.

```
- (void)setInteger:(NSInteger)value forKey:(NSString *)defaultName
```

Parameters

value
The integer value to store in the defaults database. This value is converted to an `NSString` object before being stored in the defaults database.

defaultName
The key with which to associate with the value.

Discussion
Invokes `setObject:forKey:` (page 1195) as part of its implementation.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- integerForKey:` (page 1189)

Declared In
`NSUserDefaults.h`

setObject:forKey:

Sets the value of the specified default key in the standard application domain.

```
- (void)setObject:(id)value forKey:(NSString *)defaultName
```

Parameters

value
The object to store in the defaults database. A default's value can be only property list objects: NSData, NSString, NSNumber, NSDate, NSArray, or NSDictionary.

defaultName
The key with which to associate with the value.

Discussion
Setting a default has no effect on the value returned by the `objectForKey:` (page 1189) method if the same key exists in a domain that precedes the application domain in the search list.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeObjectForKey:` (page 1192)

Declared In
`NSUserDefaults.h`

setPersistentDomain:forName:

Sets the dictionary for the specified persistent domain.

```
- (void)setPersistentDomain:(NSDictionary *)domain forName:(NSString *)domainName
```

Parameters

domain
The dictionary of keys and values you want to assign to the domain.

domainName
The domain whose keys and values you want to set. This value should be equal to your application's bundle identifier.

Discussion
When a persistent domain is changed, an `NSUserDefaultsDidChangeNotification` (page 1200) is posted.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `persistentDomainForName:` (page 1191)
- `persistentDomainNames` (page 1191)

Declared In
`NSUserDefaults.h`

setVolatileDomain:forName:

Sets the dictionary for the specified volatile domain.

```
- (void)setVolatileDomain:(NSDictionary *)domain forName:(NSString *)domainName
```

Parameters

domain
The dictionary of keys and values you want to assign to the domain.

domainName
The domain whose keys and values you want to set.

Discussion
This method raises an `NSInvalidArgumentException` if a volatile domain with the specified name already exists.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `volatileDomainForName:` (page 1199)
- `volatileDomainNames` (page 1199)

Declared In
`NSUserDefaults.h`

stringArrayForKey:

Returns the array of strings associated with the specified key.

```
- (NSArray *)stringArrayForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The array of `NSString` objects, or `nil` if the specified default does not exist, the default does not contain an array, or the array does not contain `NSString` objects.

Special Considerations
The returned array and its contents are immutable, even if the values you originally set were mutable.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `floatForKey:` (page 1187)
- `integerForKey:` (page 1189)
- `objectForKey:` (page 1189)
- `stringForKey:` (page 1197)

Declared In
`NSUserDefaults.h`

stringForKey:

Returns the string associated with the specified key.

```
- (NSString *)stringForKey:(NSString *)defaultName
```

Parameters

defaultName

A key in the current user's defaults database.

Return Value

The string associated with the specified key, or `nil` if the default does not exist or does not contain a string.

Special Considerations

The returned string is immutable, even if the value you originally set was a mutable string.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `arrayForKey:` (page 1184)
- `boolForKey:` (page 1185)
- `dataForKey:` (page 1185)
- `dictionaryForKey:` (page 1186)
- `floatForKey:` (page 1187)
- `integerForKey:` (page 1189)
- `objectForKey:` (page 1189)
- `stringArrayForKey:` (page 1197)

Declared In

`NSUserDefaults.h`

synchronize

Writes any modifications to the persistent domains to disk and updates all unmodified persistent domains to what is on disk.

```
- (BOOL)synchronize
```

Return Value

`YES` if the data was saved successfully to disk, otherwise `NO`.

Discussion

Because this method is automatically invoked at periodic intervals, use this method only if you cannot wait for the automatic synchronization (for example, if your application is about to exit) or if you want to update the user defaults to what is on disk even though you have not made any changes.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `persistentDomainForName:` (page 1191)
- `persistentDomainNames` (page 1191)
- `removePersistentDomainForName:` (page 1193)
- `setPersistentDomain:forName:` (page 1196)

Declared In
NSUserDefaults.h

volatileDomainForName:

Returns the dictionary for the specified volatile domain.

```
- (NSDictionary *)volatileDomainForName:(NSString *)domainName
```

Parameters

domainName
The domain whose keys and values you want.

Return Value
The dictionary of keys and values belonging to the domain. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property list object (`NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`).

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeVolatileDomainForName:` (page 1193)
- `setVolatileDomain:forName:` (page 1196)

Declared In
NSUserDefaults.h

volatileDomainNames

Returns an array of the current volatile domain names.

```
- (NSArray *)volatileDomainNames
```

Return Value
An array of `NSString` objects with the volatile domain names.

Discussion
You can get the contents of each domain by passing the returned domain names to the `volatileDomainForName:` (page 1199) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `removeVolatileDomainForName:` (page 1193)
- `setVolatileDomain:forName:` (page 1196)

Declared In
NSUserDefaults.h

Constants

NSUserDefaults Domains

These constants specify various user defaults domains.

```
extern NSString *NSGlobalDomain;
extern NSString *NSArgumentDomain;
extern NSString *NSRegistrationDomain;
```

Constants

`NSGlobalDomain`

The domain consisting of defaults meant to be seen by all applications.

Available in iPhone OS 2.0 and later.

Declared in `NSUserDefaults.h`

`NSArgumentDomain`

The domain consisting of defaults parsed from the application's arguments. These are one or more pairs of the form *-default value* included in the command-line invocation of the application.

Available in iPhone OS 2.0 and later.

Declared in `NSUserDefaults.h`

`NSRegistrationDomain`

The domain consisting of a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful.

Available in iPhone OS 2.0 and later.

Declared in `NSUserDefaults.h`

Declared In

`NSUserDefaults.h`

Notifications

NSUserDefaultsDidChangeNotification

This notification is posted when a change is made to defaults in a persistent domain.

The notification object is the `NSUserDefaults` object. This notification does not contain a *`userInfo`* dictionary.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSUserDefaults.h`

NSValue Class Reference

Inherits from:	NSObject
Conforms to:	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSValue.h Foundation/NSGeometry.h Foundation/NSRange.h
Companion guide:	Number and Value Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

An `NSValue` object is a simple container for a single C or Objective-C data item. It can hold any of the scalar types such as `int`, `float`, and `char`, as well as pointers, structures, and object `id`s. The purpose of this class is to allow items of such data types to be added to collections such as instances of `NSArray` and `NSSet`, which require their elements to be objects. `NSValue` objects are always immutable.

Adopted Protocols

NSCoding
- encodeWithCoder: (page 1246)
- initWithCoder: (page 1246)

NSCopying
- - copyWithZone: (page 1250)

Tasks

Creating an NSValue

- initWithBytes:objCType: (page 1206)
  Initializes and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.
+ valueWithBytes:objCType: (page 1203)
  Creates and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.
+ value:withObjCType: (page 1203)
  Creates and returns an NSValue object that contains a given value which is interpreted as being of a given Objective-C type.
+ valueWithNonretainedObject: (page 1204)
  Creates and returns an NSValue object that contains a given object.
+ valueWithPointer: (page 1204)
  Creates and returns an NSValue object that contains a given pointer.
+ valueWithRange: (page 1205)
  Creates and returns an NSValue object that contains a given NSRange structure.

Accessing Data

- getValue: (page 1205)
  Copies the receiver's value into a given buffer.
- nonretainedObjectValue (page 1207)
  Returns the receiver's value as an id.
- objCType (page 1207)
  Returns a C string containing the Objective-C type of the data contained in the receiver.
- pointerValue (page 1207)
  Returns the receiver's value as a pointer to void.
- rangeValue (page 1208)
  Returns an NSRange structure representation of the receiver.

Comparing Objects

- isEqualToValue: (page 1206)
  Returns a Boolean value that indicates whether the receiver and another value are equal.

Class Methods

value:withObjCType:

Creates and returns an NSValue object that contains a given value which is interpreted as being of a given Objective-C type.

```
+ (NSValue *)value:(const void *)value withObjCType:(const char *)type
```

Parameters

value
  The value for the new NSValue object.

type
  The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value

A new NSValue object that contains *value*, which is interpreted as being of the Objective-C type *type*.

Discussion

This method has the same effect as valueWithBytes:objCType: (page 1203) and may be deprecated in a future release. You should use valueWithBytes:objCType: (page 1203) instead.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ valueWithBytes:objCType: (page 1203)

Declared In

NSValue.h

valueWithBytes:objCType:

Creates and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.

```
+ (NSValue *)valueWithBytes:(const void *)value objCType:(const char *)type
```

Parameters

value
  The value for the new NSValue object.

type
  The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value
A new `NSValue` object that contains *`value`*, which is interpreted as being of the Objective-C type *`type`*.

Discussion
See *Number and Value Programming Topics for Cocoa* for other considerations in creating an `NSValue` object and code examples.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithBytes:objCType:` (page 1206)

Declared In
`NSValue.h`

valueWithNonretainedObject:

Creates and returns an `NSValue` object that contains a given object.

```
+ (NSValue *)valueWithNonretainedObject:(id)anObject
```

Parameters

`anObject`
The value for the new object.

Return Value
A new `NSValue` object that contains *`anObject`*.

Discussion
This method is equivalent to invoking `value:withObjCType:` (page 1203) in this manner:

```
NSValue *theValue = [NSValue value:&anObject withObjCType:@encode(void *)];
```

This method is useful for preventing an object from being retained when it's added to a collection object (such as an instance of `NSArray` or `NSDictionary`).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- nonretainedObjectValue` (page 1207)

Declared In
`NSValue.h`

valueWithPointer:

Creates and returns an `NSValue` object that contains a given pointer.

```
+ (NSValue *)valueWithPointer:(const void *)aPointer
```

Parameters

aPointer

The value for the new object.

Return Value

A new `NSValue` object that contains *aPointer*.

Discussion

This method is equivalent to invoking `value:withObjCType:` (page 1203) in this manner:

```
NSValue *theValue = [NSValue value:&aPointer withObjCType:@encode(void *)];
```

This method does not copy the contents of *aPointer*, so you must not to deallocate the memory at the pointer destination while the `NSValue` object exists. `NSData` objects may be more suited for arbitrary pointers than `NSValue` objects.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- pointerValue` (page 1207)

Declared In

`NSValue.h`

valueWithRange:

Creates and returns an `NSValue` object that contains a given `NSRange` structure.

```
+ (NSValue *)valueWithRange:(NSRange)range
```

Parameters

range

The value for the new object.

Return Value

A new `NSValue` object that contains the value of *range*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- rangeValue` (page 1208)

Declared In

`NSRange.h`

Instance Methods

getValue:

Copies the receiver's value into a given buffer.

```
- (void)getValue:(void *)buffer
```

Parameters

buffer

A buffer into which to copy the receiver's value. *buffer* must be large enough to hold the value.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSValue.h

initWithBytes:objCType:

Initializes and returns an NSValue object that contains a given value, which is interpreted as being of a given Objective-C type.

```
- (id)initWithBytes:(const void *)value objCType:(const char *)type
```

Parameters

value

The value for the new NSValue object.

type

The Objective-C type of *value*. *type* should be created with the Objective-C @encode() compiler directive; it should not be hard-coded as a C string.

Return Value

An initialized NSValue object that contains *value*, which is interpreted as being of the Objective-C type *type*. The returned object might be different than the original receiver.

Discussion

See *Number and Value Programming Topics for Cocoa* for other considerations in creating an NSValue object.

This is the designated initializer for the NSValue class.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSValue.h

isEqualToValue:

Returns a Boolean value that indicates whether the receiver and another value are equal.

```
- (BOOL)isEqualToValue:(NSValue *)value
```

Parameters

aValue

The value with which to compare the receiver.

Return Value
YES if the receiver and *aValue* are equal, otherwise NO. For NSValue objects, the class, type, and contents are compared to determine equality.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

nonretainedObjectValue

Returns the receiver's value as an id.

```
- (id)nonretainedObjectValue
```

Return Value
The receiver's value as an id. If the receiver was not created to hold a pointer-sized data item, the result is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
- getValue: (page 1205)

Declared In
NSValue.h

objCType

Returns a C string containing the Objective-C type of the data contained in the receiver.

```
- (const char *)objCType
```

Return Value
A C string containing the Objective-C type of the data contained in the receiver, as encoded by the @encode() compiler directive.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSValue.h

pointerValue

Returns the receiver's value as a pointer to void.

```
- (void *)pointerValue
```

Return Value
The receiver's value as a pointer to `void`. If the receiver was not created to hold a pointer-sized data item, the result is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- getValue:` (page 1205)

Declared In
`NSValue.h`

rangeValue

Returns an `NSRange` structure representation of the receiver.

```
- (NSRange)rangeValue
```

Return Value
An `NSRange` structure representation of the receiver.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ valueWithRange:` (page 1205)

Declared In
`NSRange.h`

NSXMLParser Class Reference

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSXMLParser.h
Companion guide:	Event-Driven XML Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

Instances of this class parse XML documents (including DTD declarations) in an event-driven manner. An `NSXMLParser` notifies its delegate about the items (elements, attributes, CDATA blocks, comments, and so on) that it encounters as it processes an XML document. It does not itself do anything with those parsed items except report them. It also reports parsing errors. For convenience, an `NSXMLParser` object in the following descriptions is sometimes referred to as a parser object.

Note: Namespace support was implemented in `NSXMLParser` for Mac OS X v10.4. Namespace-related methods of `NSXMLParser` prior to this version have no effect.

Tasks

Initializing a Parser Object

- `initWithContentsOfURL:` (page 1213)
  Initializes the receiver with the XML content referenced by the given URL.
- `initWithData:` (page 1214)
  Initializes the receiver with the XML contents encapsulated in a given data object.

Managing Delegates

- `setDelegate:` (page 1216)
  Sets the receiver's delegate.
- `delegate` (page 1213)
  Returns the receiver's delegate.

Managing Parser Behavior

- `setShouldProcessNamespaces:` (page 1216)
  Specifies whether the receiver reports the namespace and the qualified name of an element in related delegation methods .
- `shouldProcessNamespaces` (page 1217)
  Indicates whether the receiver reports the namespace and the qualified name of an element in related delegation methods.
- `setShouldReportNamespacePrefixes:` (page 1216)
  Specifies whether the receiver reports the scope of namespace declarations using related delegation methods.
- `shouldReportNamespacePrefixes` (page 1218)
  Indicates whether the receiver reports the prefixes indicating the scope of namespace declarations using related delegation methods.
- `setShouldResolveExternalEntities:` (page 1217)
  Specifies whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224).
- `shouldResolveExternalEntities` (page 1218)
  Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224).

Parsing

- `parse` (page 1215)
  Starts the event-driven parsing operation.
- `abortParsing` (page 1212)
  Stops the parser object.
- `parserError` (page 1215)
  Returns an `NSError` object from which you can obtain information about a parsing error.

Handling XML

- `parserDidStartDocument:` (page 1229) *delegate method*
  Sent by the parser object to the delegate when it begins parsing a document.
- `parserDidEndDocument:` (page 1229) *delegate method*
  Sent by the parser object to the delegate when it has successfully completed parsing.
- `parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 1220) *delegate method*
  Sent by a parser object to its delegate when it encounters a start tag for a given element.
- `parser:didEndElement:namespaceURI:qualifiedName:` (page 1219) *delegate method*
  Sent by a parser object to its delegate when it encounters an end tag for a specific element.
- `parser:didStartMappingPrefix:toURI:` (page 1221) *delegate method*
  Sent by a parser object to its delegate the first time it encounters a given namespace prefix, which is mapped to a URI.
- `parser:didEndMappingPrefix:` (page 1220) *delegate method*
  Sent by a parser object to its delegate when a given namespace prefix goes out of scope.
- `parser:resolveExternalEntityName:systemID:` (page 1227) *delegate method*
  Sent by a parser object to its delegate when it encounters a given external entity with a specific system ID.
- `parser:parseErrorOccurred:` (page 1227) *delegate method*
  Sent by a parser object to its delegate when it encounters a fatal error.
- `parser:validationErrorOccurred:` (page 1228) *delegate method*
  Sent by a parser object to its delegate when it encounters a fatal validation error. `NSXMLParser` currently does not invoke this method and does not perform validation.
- `parser:foundCharacters:` (page 1222) *delegate method*
  Sent by a parser object to provide its delegate with a string representing all or part of the characters of the current element.
- `parser:foundIgnorableWhitespace:` (page 1224) *delegate method*
  Reported by a parser object to provide its delegate with a string representing all or part of the ignorable whitespace characters of the current element.
- `parser:foundProcessingInstructionWithTarget:data:` (page 1226) *delegate method*
  Sent by a parser object to its delegate when it encounters a processing instruction.
- `parser:foundComment:` (page 1223) *delegate method*
  Sent by a parser object to its delegate when it encounters a comment in the XML.
- `parser:foundCDATA:` (page 1222) *delegate method*
  Sent by a parser object to its delegate when it encounters a CDATA block.

Handling the DTD

`- parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:` (page 1221) *delegate method*
Sent by a parser object to its delegate when it encounters a declaration of an attribute that is associated with a specific element.

`- parser:foundElementDeclarationWithName:model:` (page 1223) *delegate method*
Sent by a parser object to its delegate when it encounters a declaration of an element with a given model.

`- parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224) *delegate method*
Sent by a parser object to its delegate when it encounters an external entity declaration.

`- parser:foundInternalEntityDeclarationWithName:value:` (page 1225) *delegate method*
Sent by a parser object to the delegate when it encounters an internal entity declaration.

`- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1226) *delegate method*
Sent by a parser object to its delegate when it encounters an unparsed entity declaration.

`- parser:foundNotationDeclarationWithName:publicID:systemID:` (page 1225) *delegate method*
Sent by a parser object to its delegate when it encounters a notation declaration.

Obtaining Parser State

`- columnNumber` (page 1213)
Returns the column number of the XML document being processed by the receiver.

`- lineNumber` (page 1214)
Returns the line number of the XML document being processed by the receiver.

`- publicID` (page 1215)
Returns the public identifier of the external entity referenced in the XML document.

`- systemID` (page 1219)
Returns the system identifier of the external entity referenced in the XML document.

Instance Methods

abortParsing

Stops the parser object.

```
- (void)abortParsing
```

Discussion

If you invoke this method, the delegate, if it implements `parser:parseErrorOccurred:` (page 1227), is informed of the cancelled parsing operation.

Availability

Available in iPhone OS 2.0 and later.

See Also
- parse (page 1215)
- parserError (page 1215)

Declared In
NSXMLParser.h

columnNumber

Returns the column number of the XML document being processed by the receiver.

```
- (NSInteger)columnNumber
```

Discussion
The column refers to the nesting level of the XML elements in the document. You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in iPhone OS 2.0 and later.

See Also
- lineNumber (page 1214)

Declared In
NSXMLParser.h

delegate

Returns the receiver's delegate.

```
- (id)delegate
```

Availability
Available in iPhone OS 2.0 and later.

See Also
- setDelegate: (page 1216)

Declared In
NSXMLParser.h

initWithContentsOfURL:

Initializes the receiver with the XML content referenced by the given URL.

```
- (id)initWithContentsOfURL:(NSURL *)url
```

Parameters
url

An NSURL object specifying a URL. The URL must be fully qualified and refer to a scheme that is supported by the NSURL class.

Return Value
An initialized `NSXMLParser` object or `nil` if an error occurs.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithData:` (page 1214)

Declared In
`NSXMLParser.h`

initWithData:

Initializes the receiver with the XML contents encapsulated in a given data object.

```
- (id)initWithData:(NSData *)data
```

Parameters

data
An `NSData` object containing XML markup.

Return Value
An initialized `NSXMLParser` object or `nil` if an error occurs.

Discussion
This method is the designated initializer.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- initWithContentsOfURL:` (page 1213)

Declared In
`NSXMLParser.h`

lineNumber

Returns the line number of the XML document being processed by the receiver.

```
- (NSInteger)lineNumber
```

Discussion
You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- columnNumber` (page 1213)

Declared In
`NSXMLParser.h`

parse

Starts the event-driven parsing operation.

```
- (BOOL)parse
```

Return Value

`YES` if parsing is successful and `NO` in there is an error or if the parsing operation is aborted.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `abortParsing` (page 1212)
- `parserError` (page 1215)

Declared In

`NSXMLParser.h`

parserError

Returns an `NSError` object from which you can obtain information about a parsing error.

```
- (NSError *)parserError
```

Discussion

You may invoke this method after a parsing operation abnormally terminates to determine the cause of error.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `abortParsing` (page 1212)
- `parse` (page 1215)

Declared In

`NSXMLParser.h`

publicID

Returns the public identifier of the external entity referenced in the XML document.

```
- (NSString *)publicID
```

Discussion

You may invoke this method once a parsing operation has begun or after an error occurs.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `systemID` (page 1219)

Declared In
NSXMLParser.h

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id)delegate
```

Parameters

delegate
An object that is the new delegate. It is not retained.

Availability
Available in iPhone OS 2.0 and later.

See Also
- delegate (page 1213)

Declared In
NSXMLParser.h

setShouldProcessNamespaces:

Specifies whether the receiver reports the namespace and the qualified name of an element in related delegation methods .

```
- (void)setShouldProcessNamespaces:(BOOL)shouldProcessNamespaces
```

Parameters

shouldProcessNamespaces
YES if the receiver should report the namespace and qualified name of each element, NO otherwise. The default value is NO.

Discussion
The invoked delegation methods are parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 1220) and parser:didEndElement:namespaceURI:qualifiedName: (page 1219).

Availability
Available in iPhone OS 2.0 and later.

See Also
- shouldProcessNamespaces (page 1217)

Declared In
NSXMLParser.h

setShouldReportNamespacePrefixes:

Specifies whether the receiver reports the scope of namespace declarations using related delegation methods.

```
- (void)setShouldReportNamespacePrefixes:(BOOL)shouldReportNamespacePrefixes
```

Parameters

shouldReportNamespacePrefixes

YES if the receiver should report the scope of namespace declarations, NO otherwise. The default value is NO.

Discussion

The invoked delegation methods are parser:didStartMappingPrefix:toURI: (page 1221) and parser:didEndMappingPrefix: (page 1220).

Availability

Available in iPhone OS 2.0 and later.

See Also

- shouldReportNamespacePrefixes (page 1218)

Declared In

NSXMLParser.h

setShouldResolveExternalEntities:

Specifies whether the receiver reports declarations of external entities using the delegate method parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 1224).

```
- (void)setShouldResolveExternalEntities:(BOOL)shouldResolveExternalEntities
```

Parameters

shouldResolveExternalEntities

YES if the receiver should report declarations of external entities, NO otherwise. The default value is NO.

Discussion

If you pass in YES, you may cause other I/O operations, either network-based or disk-based, to load the external DTD.

Availability

Available in iPhone OS 2.0 and later.

See Also

- shouldResolveExternalEntities (page 1218)

Declared In

NSXMLParser.h

shouldProcessNamespaces

Indicates whether the receiver reports the namespace and the qualified name of an element in related delegation methods.

```
- (BOOL)shouldProcessNamespaces
```

Return Value

YES if the receiver reports namespace and qualified name, NO otherwise.

Discussion
The invoked delegation methods are `parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 1220) and `parser:didEndElement:namespaceURI:qualifiedName:` (page 1219).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setShouldProcessNamespaces:` (page 1216)

Declared In
`NSXMLParser.h`

shouldReportNamespacePrefixes

Indicates whether the receiver reports the prefixes indicating the scope of namespace declarations using related delegation methods.

```
- (BOOL)shouldReportNamespacePrefixes
```

Return Value
`YES` if the receiver reports the scope of namespace declarations, `NO` otherwise. The default value is `NO`.

Discussion
The invoked delegation methods are `parser:didStartMappingPrefix:toURI:` (page 1221) and `parser:didEndMappingPrefix:` (page 1220).

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setShouldReportNamespacePrefixes:` (page 1216)

Declared In
`NSXMLParser.h`

shouldResolveExternalEntities

Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224).

```
- (BOOL)shouldResolveExternalEntities
```

Return Value
`YES` if the receiver reports declarations of external entities, `NO` otherwise. The default value is `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setShouldResolveExternalEntities:` (page 1217)

Declared In
NSXMLParser.h

systemID

Returns the system identifier of the external entity referenced in the XML document.

```
- (NSString *)systemID
```

Discussion
You may invoke this method once a parsing operation has begun or after an error occurs.

Availability
Available in iPhone OS 2.0 and later.

See Also
- publicID (page 1215)

Declared In
NSXMLParser.h

Delegate Methods

parser:didEndElement:namespaceURI:qualifiedName:

Sent by a parser object to its delegate when it encounters an end tag for a specific element.

```
- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
```

Parameters

parser
A parser object.

elementName
A string that is the name of an element (in its end tag).

namespaceURI
If namespace processing is turned on, contains the URI for the current namespace as a string object.

qName
If namespace processing is turned on, contains the qualified name for the current namespace as a string object..

Availability
Available in iPhone OS 2.0 and later.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 1220)
- setShouldProcessNamespaces: (page 1216)

Declared In
NSXMLParser.h

parser:didEndMappingPrefix:

Sent by a parser object to its delegate when a given namespace prefix goes out of scope.

```
- (void)parser:(NSXMLParser *)parser didEndMappingPrefix:(NSString *)prefix
```

Parameters

parser
A parser object.

prefix
A string that is a namespace prefix.

Discussion
The parser sends this message only when namespace-prefix reporting is turned on through the `setShouldReportNamespacePrefixes:` (page 1216) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- parser:didStartMappingPrefix:toURI:` (page 1221)

Declared In
NSXMLParser.h

parser:didStartElement:namespaceURI:qualifiedName:attributes:

Sent by a parser object to its delegate when it encounters a start tag for a given element.

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qualifiedName
     attributes:(NSDictionary *)attributeDict
```

Parameters

parser
A parser object.

elementName
A string that is the name of an element (in its start tag).

namespaceURI
If namespace processing is turned on, contains the URI for the current namespace as a string object.

qualifiedName
If namespace processing is turned on, contains the qualified name for the current namespace as a string object..

attributeDict
A dictionary that contains any attributes associated with the element. Keys are the names of attributes, and values are attribute values.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `parser:didEndElement:namespaceURI:qualifiedName:` (page 1219)
- `setShouldProcessNamespaces:` (page 1216)

Declared In
`NSXMLParser.h`

parser:didStartMappingPrefix:toURI:
Sent by a parser object to its delegate the first time it encounters a given namespace prefix, which is mapped to a URI.

```
- (void)parser:(NSXMLParser *)parser didStartMappingPrefix:(NSString *)prefix
    toURI:(NSString *)namespaceURI
```

Parameters
parser
A parser object.

prefix
A string that is a namespace prefix.

namespaceURI
A string that specifies a namespace URI.

Discussion
The parser object sends this message only when namespace-prefix reporting is turned on through the `setShouldReportNamespacePrefixes:` (page 1216) method.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `parser:didEndMappingPrefix:` (page 1220)

Declared In
`NSXMLParser.h`

parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:
Sent by a parser object to its delegate when it encounters a declaration of an attribute that is associated with a specific element.

```
- (void)parser:(NSXMLParser *)parser foundAttributeDeclarationWithName:(NSString
    *)attributeName forElement:(NSString *)elementName type:(NSString *)type
    defaultValue:(NSString *)defaultValue
```

Parameters
parser
An `NSXMLParser` object parsing XML.

attributeName
A string that is the name of an attribute.

elementName
A string that is the name of an element that has the attribute *attributeName*.

type
A string, such as "ENTITY", "NOTATION", or "ID", that indicates the type of the attribute.

defaultValue
A string that specifies the default value of the attribute.

Availability
Available in iPhone OS 2.0 and later.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 1220)

Declared In
NSXMLParser.h

parser:foundCDATA:

Sent by a parser object to its delegate when it encounters a CDATA block.

```
- (void)parser:(NSXMLParser *)parser foundCDATA:(NSData *)CDATABlock
```

Parameters

parser
An NSXMLParser object parsing XML.

CDATABlock
A data object containing a block of CDATA.

Discussion
Through this method the parser object passes the contents of the block to its delegate in an NSData object. The CDATA block is character data that is ignored by the parser. The encoding of the character data is UTF-8. To convert the data object to a string object, use the NSString method initWithData:encoding: (page 1004).

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSXMLParser.h

parser:foundCharacters:

Sent by a parser object to provide its delegate with a string representing all or part of the characters of the current element.

```
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
```

Parameters

parser
A parser object.

string
A string representing the complete or partial textual content of the current element.

Discussion
The parser object may send the delegate several `parser:foundCharacters:` messages to report the characters of an element. Because *string* may be only part of the total character content for the current element, you should append it to the current accumulation of characters until the element changes.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSXMLParser.h`

parser:foundComment:

Sent by a parser object to its delegate when it encounters a comment in the XML.

```
- (void)parser:(NSXMLParser *)parser foundComment:(NSString *)comment
```

Parameters

parser
An `NSXMLParser` object parsing XML.

comment
A string that is a the content of a comment in the XML.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSXMLParser.h`

parser:foundElementDeclarationWithName:model:

Sent by a parser object to its delegate when it encounters a declaration of an element with a given model.

```
- (void)parser:(NSXMLParser *)parser foundElementDeclarationWithName:(NSString
    *)elementName model:(NSString *)model
```

Parameters

parser
An `NSXMLParser` object parsing XML.

elementName
A string that is the name of an element.

model
A string that specifies a model for *elementName*.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 1220)

Declared In
NSXMLParser.h

parser:foundExternalEntityDeclarationWithName:publicID:systemID:

Sent by a parser object to its delegate when it encounters an external entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundExternalEntityDeclarationWithName:(NSString
    *)entityName publicID:(NSString *)publicID systemID:(NSString *)systemID
```

Parameters

parser
An NSXMLParser object parsing XML.

entityName
A string that is the name of an entity.

publicID
A string that specifies the public ID associated with *entityName*.

systemID
A string that specifies the system ID associated with *entityName*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- parser:foundInternalEntityDeclarationWithName:value: (page 1225)
- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName: (page 1226)
- parser:resolveExternalEntityName:systemID: (page 1227)

Declared In
NSXMLParser.h

parser:foundIgnorableWhitespace:

Reported by a parser object to provide its delegate with a string representing all or part of the ignorable whitespace characters of the current element.

```
- (void)parser:(NSXMLParser *)parser foundIgnorableWhitespace:(NSString
    *)whitespaceString
```

Parameters

parser
A parser object.

whitespaceString
A string representing all or part of the ignorable whitespace characters of the current element.

Discussion

All the whitespace characters of the element (including carriage returns, tabs, and new-line characters) may not be provided through an individual invocation of this method. The parser may send the delegate several `parser:foundIgnorableWhitespace:` messages to report the whitespace characters of an element. You should append the characters in each invocation to the current accumulation of characters.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- parser:foundCharacters:` (page 1222)

Declared In

`NSXMLParser.h`

parser:foundInternalEntityDeclarationWithName:value:

Sent by a parser object to the delegate when it encounters an internal entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundInternalEntityDeclarationWithName:(NSString
    *)name value:(NSString *)value
```

Parameters

parser

An `NSXMLParser` object parsing XML.

name

A string that is the declared name of an internal entity.

value

A string that is the value of entity *name*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224)

`- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1226)

Declared In

`NSXMLParser.h`

parser:foundNotationDeclarationWithName:publicID:systemID:

Sent by a parser object to its delegate when it encounters a notation declaration.

```
- (void)parser:(NSXMLParser *)parser foundNotationDeclarationWithName:(NSString
    *)name publicID:(NSString *)publicID systemID:(NSString *)systemID
```

Parameters

parser

An `NSXMLParser` object parsing XML.

name
A string that is the name of the notation.

publicID
A string specifying the public ID associated with the notation *name*.

systemID
A string specifying the system ID associated with the notation *name*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSXMLParser.h

parser:foundProcessingInstructionWithTarget:data:

Sent by a parser object to its delegate when it encounters a processing instruction.

```
- (void)parser:(NSXMLParser *)parser foundProcessingInstructionWithTarget:(NSString
    *)target data:(NSString *)data
```

Parameters

parser
A parser object.

target
A string representing the target of a processing instruction.

data
A string representing the data for a processing instruction.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSXMLParser.h

parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:

Sent by a parser object to its delegate when it encounters an unparsed entity declaration.

```
- (void)parser:(NSXMLParser *)parser foundUnparsedEntityDeclarationWithName:(NSString
    *)name publicID:(NSString *)publicID systemID:(NSString *)systemID
   notationName:(NSString *)notationName
```

Parameters

parser
An `NSXMLParser` object parsing XML.

name
A string that is the name of the unparsed entity in the declaration.

publicID
A string specifying the public ID associated with the entity *name*.

systemID
A string specifying the system ID associated with the entity *name*.

notationName
A string specifying a notation of the declaration of entity *name*.

Availability
Available in iPhone OS 2.0 and later.

See Also
- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 1224)
- parser:foundInternalEntityDeclarationWithName:value: (page 1225)
- parser:resolveExternalEntityName:systemID: (page 1227)

Declared In
NSXMLParser.h

parser:parseErrorOccurred:

Sent by a parser object to its delegate when it encounters a fatal error.

```
- (void)parser:(NSXMLParser *)parser parseErrorOccurred:(NSError *)parseError
```

Parameters

parser
A parser object.

parseError
An NSError object describing the parsing error that occurred.

Discussion
When this method is invoked, parsing is stopped. For further information about the error, you can query *parseError* or you can send the receiver a parserError (page 1215) message. You can also send the parser lineNumber (page 1214) and columnNumber (page 1213) messages to further isolate where the error occurred. Typically you implement this method to display information about the error to the user.

Availability
Available in iPhone OS 2.0 and later.

See Also
- parser:validationErrorOccurred: (page 1228)

Declared In
NSXMLParser.h

parser:resolveExternalEntityName:systemID:

Sent by a parser object to its delegate when it encounters a given external entity with a specific system ID.

```
- (NSData *)parser:(NSXMLParser *)parser resolveExternalEntityName:(NSString
    *)entityName systemID:(NSString *)systemID
```

Parameters

parser
A parser object.

entityName
A string that specifies the external name of an entity.

systemID
A string that specifies the system ID for the external entity.

Return Value

An `NSData` object that contains the resolution of the given external entity.

Discussion

The delegate can resolve the external entity (for example, locating and reading an externally declared DTD) and provide the result to the parser object as an `NSData` object.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1224)
- `parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1226)

Declared In

`NSXMLParser.h`

parser:validationErrorOccurred:

Sent by a parser object to its delegate when it encounters a fatal validation error. `NSXMLParser` currently does not invoke this method and does not perform validation.

```
- (void)parser:(NSXMLParser *)parser validationErrorOccurred:(NSError *)validError
```

Parameters

parser
A parser object.

validError
An `NSError` object describing the validation error that occurred.

Discussion

If you want to validate an XML document, use the validation features of the `NSXMLDocument` class.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `parser:parseErrorOccurred:` (page 1227)

Declared In

`NSXMLParser.h`

parserDidEndDocument:

Sent by the parser object to the delegate when it has successfully completed parsing.

```
- (void)parserDidEndDocument:(NSXMLParser *)parser
```

Parameters

parser
A parser object.

Availability

Available in iPhone OS 2.0 and later.

See Also

- parserDidStartDocument: (page 1229)

Declared In

NSXMLParser.h

parserDidStartDocument:

Sent by the parser object to the delegate when it begins parsing a document.

```
- (void)parserDidStartDocument:(NSXMLParser *)parser
```

Parameters

parser
A parser object.

Availability

Available in iPhone OS 2.0 and later.

See Also

- parserDidEndDocument: (page 1229)

Declared In

NSXMLParser.h

Constants

NSXMLParserErrorDomain

This constant defines the NSXMLParser error domain.

```
NSString * const NSXMLParserErrorDomain
```

Constants

`NSXMLParserErrorDomain`

Indicates an error in XML parsing.

Used by `NSError`.

Available in iPhone OS 2.0 and later.

Declared in `NSXMLParser.h`

Declared In

`NSXMLParser.h`

NSXMLParserError

A type defined for the contants listed in "Parser Error Constants" (page 1230).

```
typedef NSInteger NSXMLParserError;
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSXMLParser.h`

Parser Error Constants

The following error types are defined by `NSXMLParser`.

```
typedef enum {
    NSXMLParserInternalError = 1,
    NSXMLParserOutOfMemoryError = 2,
    NSXMLParserDocumentStartError = 3,
    NSXMLParserEmptyDocumentError = 4,
    NSXMLParserPrematureDocumentEndError = 5,
    NSXMLParserInvalidHexCharacterRefError = 6,
    NSXMLParserInvalidDecimalCharacterRefError = 7,
    NSXMLParserInvalidCharacterRefError = 8,
    NSXMLParserInvalidCharacterError = 9,
    NSXMLParserCharacterRefAtEOFError = 10,
    NSXMLParserCharacterRefInPrologError = 11,
    NSXMLParserCharacterRefInEpilogError = 12,
    NSXMLParserCharacterRefInDTDError = 13,
    NSXMLParserEntityRefAtEOFError = 14,
    NSXMLParserEntityRefInPrologError = 15,
    NSXMLParserEntityRefInEpilogError = 16,
    NSXMLParserEntityRefInDTDError = 17,
    NSXMLParserParsedEntityRefAtEOFError = 18,
    NSXMLParserParsedEntityRefInPrologError = 19,
    NSXMLParserParsedEntityRefInEpilogError = 20,
    NSXMLParserParsedEntityRefInInternalSubsetError = 21,
    NSXMLParserEntityReferenceWithoutNameError = 22,
    NSXMLParserEntityReferenceMissingSemiError = 23,
    NSXMLParserParsedEntityRefNoNameError = 24,
    NSXMLParserParsedEntityRefMissingSemiError = 25,
    NSXMLParserUndeclaredEntityError = 26,
    NSXMLParserUnparsedEntityError = 28,
    NSXMLParserEntityIsExternalError = 29,
    NSXMLParserEntityIsParameterError = 30,
    NSXMLParserUnknownEncodingError = 31,
    NSXMLParserEncodingNotSupportedError = 32,
    NSXMLParserStringNotStartedError = 33,
    NSXMLParserStringNotClosedError = 34,
    NSXMLParserNamespaceDeclarationError = 35,
    NSXMLParserEntityNotStartedError = 36,
    NSXMLParserEntityNotFinishedError = 37,
    NSXMLParserLessThanSymbolInAttributeError = 38,
    NSXMLParserAttributeNotStartedError = 39,
    NSXMLParserAttributeNotFinishedError = 40,
    NSXMLParserAttributeHasNoValueError = 41,
    NSXMLParserAttributeRedefinedError = 42,
    NSXMLParserLiteralNotStartedError = 43,
    NSXMLParserLiteralNotFinishedError = 44,
    NSXMLParserCommentNotFinishedError = 45,
    NSXMLParserProcessingInstructionNotStartedError = 46,
    NSXMLParserProcessingInstructionNotFinishedError = 47,
    NSXMLParserNotationNotStartedError = 48,
    NSXMLParserNotationNotFinishedError = 49,
    NSXMLParserAttributeListNotStartedError = 50,
    NSXMLParserAttributeListNotFinishedError = 51,
    NSXMLParserMixedContentDeclNotStartedError = 52,
    NSXMLParserMixedContentDeclNotFinishedError = 53,
    NSXMLParserElementContentDeclNotStartedError = 54,
    NSXMLParserElementContentDeclNotFinishedError = 55,
    NSXMLParserXMLDeclNotStartedError = 56,
    NSXMLParserXMLDeclNotFinishedError = 57,
    NSXMLParserConditionalSectionNotStartedError = 58,
```

```
    NSXMLParserConditionalSectionNotFinishedError = 59,
    NSXMLParserExternalSubsetNotFinishedError = 60,
    NSXMLParserDOCTYPEDeclNotFinishedError = 61,
    NSXMLParserMisplacedCDATAEndStringError = 62,
    NSXMLParserCDATANotFinishedError = 63,
    NSXMLParserMisplacedXMLDeclarationError = 64,
    NSXMLParserSpaceRequiredError = 65,
    NSXMLParserSeparatorRequiredError = 66,
    NSXMLParserNMTOKENRequiredError = 67,
    NSXMLParserNAMERequiredError = 68,
    NSXMLParserPCDATARequiredError = 69,
    NSXMLParserURIRequiredError = 70,
    NSXMLParserPublicIdentifierRequiredError = 71,
    NSXMLParserLTRequiredError = 72,
    NSXMLParserGTRequiredError = 73,
    NSXMLParserLTSlashRequiredError = 74,
    NSXMLParserEqualExpectedError = 75,
    NSXMLParserTagNameMismatchError = 76,
    NSXMLParserUnfinishedTagError = 77,
    NSXMLParserStandaloneValueError = 78,
    NSXMLParserInvalidEncodingNameError = 79,
    NSXMLParserCommentContainsDoubleHyphenError = 80,
    NSXMLParserInvalidEncodingError = 81,
    NSXMLParserExternalStandaloneEntityError = 82,
    NSXMLParserInvalidConditionalSectionError = 83,
    NSXMLParserEntityValueRequiredError = 84,
    NSXMLParserNotWellBalancedError = 85,
    NSXMLParserExtraContentError = 86,
    NSXMLParserInvalidCharacterInEntityError = 87,
    NSXMLParserParsedEntityRefInInternalError = 88,
    NSXMLParserEntityRefLoopError = 89,
    NSXMLParserEntityBoundaryError = 90,
    NSXMLParserInvalidURIError = 91,
    NSXMLParserURIFragmentError = 92,
    NSXMLParserNoDTDError = 94,
    NSXMLParserDelegateAbortedParseError = 512
} NSXMLParserError;
```

Constants

`NSXMLParserInternalError`

The parser object encountered an internal error.

Available in iPhone OS 2.0 and later.

Declared in `NSXMLParser.h`

`NSXMLParserOutOfMemoryError`

The parser object ran out of memory.

Available in iPhone OS 2.0 and later.

Declared in `NSXMLParser.h`

`NSXMLParserDocumentStartError`

The parser object is unable to start parsing.

Available in iPhone OS 2.0 and later.

Declared in `NSXMLParser.h`

`NSXMLParserEmptyDocumentError`
The document is empty.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserPrematureDocumentEndError`
The document ended unexpectedly.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidHexCharacterRefError`
Invalid hexadecimal character reference encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidDecimalCharacterRefError`
Invalid decimal character reference encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidCharacterRefError`
Invalid character reference encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidCharacterError`
Invalid character encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCharacterRefAtEOFError`
Target of character reference cannot be found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCharacterRefInPrologError`
Invalid character found in the prolog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCharacterRefInEpilogError`
Invalid character found in the epilog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCharacterRefInDTDError`
Invalid character encountered in the DTD.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityRefAtEOFError`
Target of entity reference is not found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityRefInPrologError`
Invalid entity reference found in the prolog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityRefInEpilogError`
Invalid entity reference found in the epilog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityRefInDTDError`
Invalid entity reference found in the DTD.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefAtEOFError`
Target of parsed entity reference is not found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefInPrologError`
Target of parsed entity reference is not found in prolog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefInEpilogError`
Target of parsed entity reference is not found in epilog.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefInInternalSubsetError`
Target of parsed entity reference is not found in internal subset.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityReferenceWithoutNameError`
Entity reference is without name.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityReferenceMissingSemiError`
Entity reference is missing semicolon.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefNoNameError`
Parsed entity reference is without an entity name.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefMissingSemiError`
Parsed entity reference is missing semicolon.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserUndeclaredEntityError`
Entity is not declared.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserUnparsedEntityError`
Cannot parse entity.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityIsExternalError`
Cannot parse external entity.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityIsParameterError`
Entity is a parameter.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserUnknownEncodingError`
Document encoding is unknown.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEncodingNotSupportedError`
Document encoding is not supported.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserStringNotStartedError`
String is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserStringNotClosedError`
String is not closed.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNamespaceDeclarationError`
Invalid namespace declaration encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityNotStartedError`
Entity is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityNotFinishedError`
Entity is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserLessThanSymbolInAttributeError`
Angle bracket is used in attribute.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeNotStartedError`
Attribute is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeNotFinishedError`
Attribute is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeHasNoValueError`
Attribute doesn't contain a value.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeRedefinedError`
Attribute is redefined.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserLiteralNotStartedError`
Literal is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserLiteralNotFinishedError`
Literal is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCommentNotFinishedError`
Comment is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserProcessingInstructionNotStartedError`
Processing instruction is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserProcessingInstructionNotFinishedError`
Processing instruction is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNotationNotStartedError`
Notation is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNotationNotFinishedError`
Notation is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeListNotStartedError`
Attribute list is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserAttributeListNotFinishedError`
Attribute list is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserMixedContentDeclNotStartedError`
Mixed content declaration is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserMixedContentDeclNotFinishedError`
Mixed content declaration is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserElementContentDeclNotStartedError`
Element content declaration is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserElementContentDeclNotFinishedError`
Element content declaration is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserXMLDeclNotStartedError`
XML declaration is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserXMLDeclNotFinishedError`
XML declaration is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserConditionalSectionNotStartedError`
Conditional section is not started.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserConditionalSectionNotFinishedError`
Conditional section is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserExternalSubsetNotFinishedError`
External subset is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserDOCTYPEDeclNotFinishedError`
Document type declaration is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserMisplacedCDATAEndStringError`
Misplaced CDATA end string.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCDATANotFinishedError`
CDATA block is not finished.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserMisplacedXMLDeclarationError`
Misplaced XML declaration.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserSpaceRequiredError`
Space is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserSeparatorRequiredError`
Separator is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNMTOKENRequiredError`
Name token is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNAMERequiredError`
Name is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserPCDATARequiredError`
CDATA is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserURIRequiredError`
URI is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserPublicIdentifierRequiredError`
Public identifier is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserLTRequiredError`
Left angle bracket is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserGTRequiredError`
Right angle bracket is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserLTSlashRequiredError`
Left angle bracket slash is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEqualExpectedError`
Equal sign expected.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserTagNameMismatchError`
Tag name mismatch.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserUnfinishedTagError`
Unfinished tag found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserStandaloneValueError`
Standalone value found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidEncodingNameError`
Invalid encoding name found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserCommentContainsDoubleHyphenError`
Comment contains double hyphen.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidEncodingError`
Invalid encoding.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserExternalStandaloneEntityError`
External standalone entity.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidConditionalSectionError`
Invalid conditional section.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityValueRequiredError`
Entity value is required.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNotWellBalancedError`
Document is not well balanced.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserExtraContentError`
Error in content found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidCharacterInEntityError`
Invalid character in entity found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserParsedEntityRefInInternalError`
Internal error in parsed entity reference found.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityRefLoopError`
Entity reference loop encountered.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserEntityBoundaryError`
Entity boundary error.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserInvalidURIError`
Invalid URI specified.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserURIFragmentError`
URI fragment.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserNoDTDError`
Missing DTD.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

`NSXMLParserDelegateAbortedParseError`
Delegate aborted parse.
Available in iPhone OS 2.0 and later.
Declared in `NSXMLParser.h`

Declared In
`NSXMLParser.h`

Protocols

NSCoding Protocol Reference

Adopted by:	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSObject.h
Companion guide:	Archives and Serializations Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSCoding` protocol declares the two methods that a class must implement so that instances of that class can be encoded and decoded. This capability provides the basis for archiving (where objects and other structures are stored on disk) and distribution (where objects are copied to different address spaces).

In keeping with object-oriented design principles, an object being encoded or decoded is responsible for encoding and decoding its instance variables. A coder instructs the object to do so by invoking `encodeWithCoder:` (page 1246) or `initWithCoder:` (page 1246). `encodeWithCoder:` (page 1246) instructs the object to encode its instance variables to the coder provided; an object can receive this method any number of times. `initWithCoder:` (page 1246) instructs the object to initialize itself from data in the coder provided; as such, it replaces any other initialization method and is sent only once per object. Any object class that should be codable must adopt the `NSCoding` protocol and implement its methods.

It is important to consider the possible types of archiving that a coder supports. On Mac OS X version 10.2 and later, keyed archiving is preferred. You may, however, need to support classic archiving. For details, see *Archives and Serializations Programming Guide for Cocoa*.

Tasks

Initializing with a Coder

`- initWithCoder:` (page 1246)
Returns an object initialized from data in a given unarchiver.

Encoding with a Coder

`- encodeWithCoder:` (page 1246)
Encodes the receiver using a given archiver.

Instance Methods

encodeWithCoder:

Encodes the receiver using a given archiver.

```
- (void)encodeWithCoder:(NSCoder *)encoder
```

Parameters

encoder
An archiver object.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSObject.h`

initWithCoder:

Returns an object initialized from data in a given unarchiver.

```
- (id)initWithCoder:(NSCoder *)decoder
```

Parameters

decoder
An unarchiver object.

Return Value

`self`, initialized using the data in *decoder*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSObject.h`

NSCopying Protocol Reference

Adopted by:	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSObject.h
Companion guide:	Memory Management Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSCopying` protocol declares a method for providing functional copies of an object. The exact meaning of "copy" can vary from class to class, but a copy must be a functionally independent object with values identical to the original at the time the copy was made. A copy produced with `NSCopying` is implicitly retained by the sender, who is responsible for releasing it.

`NSCopying` declares one method, `copyWithZone:` (page 1250), but copying is commonly invoked with the convenience method `copy`. The `copy` method is defined for all objects inheriting from `NSObject` and simply invokes `copyWithZone:` (page 1250) with the default zone.

Your options for implementing this protocol are as follows:

- Implement `NSCopying` using `alloc` (page 783) and `init...` in classes that don't inherit `copyWithZone:` (page 1250).

- Implement NSCopying by invoking the superclass's copyWithZone: (page 1250) when NSCopying behavior is inherited. If the superclass implementation might use the NSCopyObject (page 1347) function, make explicit assignments to pointer instance variables for retained objects.
- Implement NSCopying by retaining the original instead of creating a new copy when the class and its contents are immutable.

If a subclass inherits NSCopying from its superclass and declares additional instance variables, the subclass has to override copyWithZone: (page 1250) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

- copyWithZone: (page 1250)
  Returns a new instance that's a copy of the receiver.

Instance Methods

copyWithZone:

Returns a new instance that's a copy of the receiver.

```
- (id)copyWithZone:(NSZone *)zone
```

Parameters

zone

The zone identifies an area of memory from which to allocate for the new instance. If *zone* is NULL, the new instance is allocated from the default zone, which is returned from the function NSDefaultMallocZone.

Discussion

The returned object is implicitly retained by the sender, who is responsible for releasing it. The copy returned is immutable if the consideration "immutable vs. mutable" applies to the receiving object; otherwise the exact nature of the copy is determined by the class.

Availability

Available in iPhone OS 2.0 and later.

See Also

- mutableCopyWithZone: (page 1300) (NSMutableCopying protocol)
- copy (page 798) (NSObject class)

Declared In

NSObject.h

NSDecimalNumberBehaviors Protocol Reference

Adopted by:	NSDecimalNumberHandler
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSDecimalNumber.h
Companion guide:	Number and Value Programming Topics for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSDecimalBehaviors` protocol declares three methods that control the discretionary aspects of working with `NSDecimalNumber` objects.

The `scale` (page 1253) and `roundingMode` (page 1253) methods determine the precision of `NSDecimalNumber`'s return values and the way in which those values should be rounded to fit that precision. The `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1252) method determines the way in which an `NSDecimalNumber` object should handle different calculation errors.

For an example of a class that adopts the `NSDecimalBehaviors` protocol, see the specification for NSDecimalNumberHandler.

Tasks

Rounding

- `roundingMode` (page 1253)
  Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values.
- `scale` (page 1253)
  Returns the number of digits allowed after the decimal separator.

Handling Errors

- `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1252)
  Specifies what an `NSDecimalNumber` object will do when it encounters an error.

Instance Methods

exceptionDuringOperation:error:leftOperand:rightOperand:

Specifies what an `NSDecimalNumber` object will do when it encounters an error.

```
- (NSDecimalNumber *)exceptionDuringOperation:(SEL)method
    error:(NSCalculationError)error leftOperand:(NSDecimalNumber *)leftOperand
    rightOperand:(NSDecimalNumber *)rightOperand
```

Parameters

method
The method that was being executed when the error occurred.

error
The type of error that was generated.

leftOperand
The left operand.

rightOperand
The right operand.

Discussion

There are four possible values for *error*, described in NSCalculationError (page 1255). The first three have to do with limits on the ability of `NSDecimalNumber` to represent decimal numbers. An `NSDecimalNumber` object can represent any number that can be expressed as mantissa x 10^exponent, where mantissa is a decimal integer up to 38 digits long, and exponent is between –256 and 256. The fourth results from the caller trying to divide by `0`.

In implementing `exceptionDuringOperation:error:leftOperand:rightOperand:`, you can handle each of these errors in several ways:

- Raise an exception. For an explanation of exceptions, see *Exception Programming Topics for Cocoa*.

- Return `nil`. The calling method will return its value as though no error had occurred. If *`error`* is `NSCalculationLossOfPrecision`, *`method`* will return an imprecise value—that is, one constrained to 38 significant digits. If *`error`* is `NSCalculationUnderflow` or `NSCalculationOverflow`, *`method`* will return `NSDecimalNumber`'s `notANumber`. You shouldn't return `nil` if *`error`* is `NSDivideByZero`.
- Correct the error and return a valid `NSDecimalNumber` object. The calling method will use this as its own return value.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

roundingMode

Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values.

```
- (NSRoundingMode)roundingMode
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

scale

Returns the number of digits allowed after the decimal separator.

```
- (short)scale
```

Return Value
The number of digits allowed after the decimal separator.

Discussion
This method limits the precision of the values returned by `NSDecimalNumber`'s `decimalNumberBy...` methods. If `scale` returns a negative value, it affects the digits before the decimal separator as well. If `scale` returns `NSDecimalNoScale`, the number of digits is unlimited.

Assuming that `roundingMode` (page 1253) returns `NSRoundPlain`, different values of `scale` have the following effects on the number 123.456:

Scale	Return Value
`NSDecimalNoScale`	123.456
2	123.45
0	123
–2	100

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimalNumber.h`

Constants

NSRoundingMode

These constants specify rounding behaviors.

```
typedef enum {
    NSRoundPlain,
    NSRoundDown,
    NSRoundUp,
    NSRoundBankers
} NSRoundingMode;
```

Constants

`NSRoundPlain`
Round to the closest possible return value; when caught halfway between two positive numbers, round up; when caught between two negative numbers, round down.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSRoundDown`
Round return values down.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSRoundUp`
Round return values up.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSRoundBankers`
Round to the closest possible return value; when halfway between two possibilities, return the possibility whose last digit is even.

In practice, this means that, over the long run, numbers will be rounded up as often as they are rounded down; there will be no systematic bias.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

Discussion
The rounding mode matters only if the `scale` (page 1253) method sets a limit on the precision of `NSDecimalNumber` return values. It has no effect if `scale` returns `NSDecimalNoScale`. Assuming that `scale` (page 1253) returns 1, the rounding mode has the following effects on various original values:

Original Value	NSRoundPlain	NSRoundDown	NSRoundUp	NSRoundBankers
1.24	1.2	1.2	1.3	1.2
1.26	1.3	1.2	1.3	1.3
1.25	1.3	1.2	1.3	1.2
1.35	1.4	1.3	1.4	1.4
–1.35	–1.4	–1.4	–1.3	–1.4

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimal.h`

NSCalculationError

Calculation error constants used to describe an error in `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1252).

```
typedef enum {
    NSCalculationNoError = 0,
    NSCalculationLossOfPrecision,
    NSCalculationUnderflow,
    NSCalculationOverflow,
    NSCalculationDivideByZero
} NSCalculationError;
```

Constants

`NSCalculationNoError`

No error occurred.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSCalculationLossOfPrecision`

The number can't be represented in 38 significant digits.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSCalculationOverflow`

The number is too large to represent.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSCalculationUnderflow`

The number is too small to represent.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSCalculationDivideByZero`

The caller tried to divide by `0`.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimal.h`

NSErrorRecoveryAttempting Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in:	Foundation/NSErrorRecoveryAttempting.h
Availability:	Available in Mac OS X v10.4 and later.

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSErrorRecoveryAttempting` informal protocol provides methods that allow your application to attempt to recover from an error. These methods are invoked when an `NSError` object is returned that specifies the implementing object as the error `recoveryAttempter` and the user has selected one of the error's localized recovery options.

Which method is invoked is dependent on how the error is presented to the user. If the error is presented in a document-modal sheet, `attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:` (page 1258) is invoked. If the error is presented in an application-modal dialog, `attemptRecoveryFromError:optionIndex:` (page 1258) is invoked.

Tasks

Attempting Recovery From Errors

- attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo: (page 1258)
  Implemented to attempt a recovery from an error noted in an document-modal sheet.
- attemptRecoveryFromError:optionIndex: (page 1258)
  Implemented to attempt a recovery from an error noted in an application-modal dialog.

Instance Methods

attemptRecoveryFromError:optionIndex:

Implemented to attempt a recovery from an error noted in an application-modal dialog.

```
- (BOOL)attemptRecoveryFromError:(NSError *)error
    optionIndex:(NSUInteger)recoveryOptionIndex
```

Parameters

error
  An NSError object that describes the error, including error recovery options.

recoveryOptionIndex
  The index of the user selected recovery option in *error*'s localized recovery array.

Return Value

YES if the error recovery was completed successfully, NO otherwise.

Discussion

Invoked when an error alert is been presented to the user in an application-modal dialog, and the user has selected an error recovery option specified by *error*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSError.h

attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:

Implemented to attempt a recovery from an error noted in an document-modal sheet.

```
- (void)attemptRecoveryFromError:(NSError *)error
    optionIndex:(NSUInteger)recoveryOptionIndex delegate:(id)delegate
    didRecoverSelector:(SEL)didRecoverSelector contextInfo:(void *)contextInfo
```

Parameters

error

An `NSError` object that describes the error, including error recovery options.

recoveryOptionIndex

The index of the user selected recovery option in *error*'s localized recovery array.

delegate

An object that is the modal delegate.

didRecoverSelector

A selector identifying the method implemented by the modal delegate.

contextInfo

Arbitrary data associated with the attempt at error recovery, to be passed to *delegate* in *didRecoverSelector*.

Discussion

Invoked when an error alert is presented to the user in a document-modal sheet, and the user has selected an error recovery option specified by *error*. After recovery is attempted, your implementation should send *delegate* the message specified in *didRecoverSelector*, passing the provided *contextInfo*.

The *didRecoverSelector* should have the following signature:

```
- (void)didPresentErrorWithRecovery:(BOOL)didRecover contextInfo:(void
*)contextInfo;
```

where *didRecover* is `YES` if the error recovery attempt was successful; otherwise it is `NO`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSError.h`

NSFastEnumeration Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSEnumerator.h
Companion guide:	The Objective-C 2.0 Programming Language

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The fast enumeration protocol `NSFastEnumeration` must be adopted and implemented by objects used in conjunction with the for language construct used in conjunction with Cocoa objects.

The abstract class `NSEnumerator` provides a convenience implementation that uses `nextObject` (page 341) to return items one at a time. For more details, see Fast Enumeration.

Tasks

Enumeration

`- countByEnumeratingWithState:objects:count:` (page 1262)
Returns by reference a C array of objects over which the sender should iterate, and as the return value the number of objects in the array.

Instance Methods

countByEnumeratingWithState:objects:count:

Returns by reference a C array of objects over which the sender should iterate, and as the return value the number of objects in the array.

```
- (NSUInteger)countByEnumeratingWithState:(NSFastEnumerationState *)state
    objects:(id *)stackbuf
    count:(NSUInteger)len
```

Parameters

state
Context information that is used in the enumeration to, in addition to other possibilities, ensure that the collection has not been mutated.

stackbuf
A C array of objects over which the sender is to iterate.

len
The maximum number of objects to return in *stackbuf*.

Return Value

The number of objects returned in *stackbuf*. Returns `0` when the iteration is finished.

Discussion

The state structure is assumed to be of stack local memory and, from a garbage collection perspective, does not require write-barriers on stores, so you can recast the passed in state structure to one more suitable for your iteration.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSEnumerator.h`

Constants

NSFastEnumerationState

This defines the structure used as contextual information in the `NSFastEnumeration` protocol.

```
typedef struct {
    unsigned long state;
    id *itemsPtr;
    unsigned long *mutationsPtr;
    unsigned long extra[5];
} NSFastEnumerationState;
```

Fields

`state`

Arbitrary state information used by the iterator. Typically this is set to `0` at the beginning of the iteration.

`itemsPtr`

A C array of objects.

`mutationsPtr`

Arbitrary state information used to detect whether the collection has been mutated.

`extra`

A C array that you can use to hold returned values.

Discussion

For more information, see `countByEnumeratingWithState:objects:count:` (page 1262).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSEnumerator.h`

NSKeyValueCoding Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in:	Foundation/NSKeyValueCoding.h
Companion guide:	Key-Value Coding Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSKeyValueCoding` informal protocol defines a mechanism by which you can access the properties of an object indirectly by name (or key), rather than directly through invocation of an accessor method or as instance variables. Thus, all of an object's properties can be accessed in a consistent manner.

The basic methods for accessing an object's values are `setValue:forKey:` (page 1272), which sets the value for the property identified by the specified key, and `valueForKey:` (page 1276), which returns the value for the property identified by the specified key. The default implementation uses the accessor methods normally implemented by objects (or to access instance variables directly if need be).

Tasks

Getting Values

- `valueForKey:` (page 1276)
  Returns the value for the property identified by a given key.
- `valueForKeyPath:` (page 1277)
  Returns the value for the derived property identified by a given key path.
- `dictionaryWithValuesForKeys:` (page 1267)
  Returns a dictionary containing the property values identified by each of the keys in a given array.
- `valueForUndefinedKey:` (page 1277)
  Invoked by `valueForKey:` (page 1276) when it finds no property corresponding to a given key.
- `mutableArrayValueForKey:` (page 1268)
  Returns a mutable array that provides read-write access to an ordered to-many relationship specified by a given key.
- `mutableArrayValueForKeyPath:` (page 1269)
  Returns a mutable array that provides read-write access to the ordered to-many relationship specified by a given key path.
- `mutableSetValueForKey:` (page 1270)
  Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key.
- `mutableSetValueForKeyPath:` (page 1271)
  Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key path.

Setting Values

- `setValue:forKeyPath:` (page 1273)
  Sets the value for the property identified by a given key path to a given value.
- `setValuesForKeysWithDictionary:` (page 1274)
  Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties.
- `setNilValueForKey:` (page 1271)
  Invoked by `setValue:forKey:` (page 1272) when it's given a `nil` value for a scalar value (such as an `int` or `float`).
- `setValue:forKey:` (page 1272)
  Sets the property of the receiver specified by a given key to a given value.
- `setValue:forUndefinedKey:` (page 1274)
  Invoked by `setValue:forKey:` (page 1272) when it finds no property for a given key.

Changing Default Behavior

`+ accessInstanceVariablesDirectly` (page 1267)
Returns a Boolean value that indicates whether the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property.

Validation

`- validateValue:forKey:error:` (page 1274)
Returns a Boolean value that indicates whether the value specified by a given pointer is valid for the property identified by a given key.

`- validateValue:forKeyPath:error:` (page 1275)
Returns a Boolean value that indicates whether the value specified by a given pointer is valid for a given key path relative to the receiver.

Class Methods

accessInstanceVariablesDirectly

Returns a Boolean value that indicates whether the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property.

```
+ (BOOL)accessInstanceVariablesDirectly
```

Return Value

`YES` if the key-value coding methods should access the corresponding instance variable directly on finding no accessor method for a property, otherwise `NO`.

Discussion

The default returns `YES`. Subclasses can override it to return `NO`, in which case the key-value coding methods won't access instance variables.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyValueCoding.h`

Instance Methods

dictionaryWithValuesForKeys:

Returns a dictionary containing the property values identified by each of the keys in a given array.

```
- (NSDictionary *)dictionaryWithValuesForKeys:(NSArray *)keys
```

Parameters

keys

An array containing `NSString` objects that identify properties of the receiver.

Return Value

A dictionary containing as keys the property names in *keys*, with corresponding values being the corresponding property values.

Discussion

The default implementation invokes `valueForKey:` (page 1276) for each key in *keys* and substitutes `NSNull` values in the dictionary for returned `nil` values.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setValuesForKeysWithDictionary:` (page 1274)

Declared In

`NSKeyValueCoding.h`

mutableArrayValueForKey:

Returns a mutable array that provides read-write access to an ordered to-many relationship specified by a given key.

```
- (NSMutableArray *)mutableArrayValueForKey:(NSString *)key
```

Parameters

key

The name of an ordered to-many relationship.

Return Value

A mutable array that provides read-write access to the ordered to-many relationship specified by *key*.

Discussion

Objects added to the mutable array become related to the receiver, and objects removed from the mutable array become unrelated. The default implementation recognizes the same simple accessor methods and array accessor methods as `valueForKey:` (page 1276), and follows the same direct instance variable access policies, but always returns a mutable collection proxy object instead of the immutable collection that `valueForKey:` would return.

This method also:

1. Searches the class of the receiver for a pair of methods whose names match the patterns `insertObject:in<Key>AtIndex:` and `removeObjectFrom<Key>AtIndex:` (and therefore correspond to the two most primitive methods defined by the `NSMutableArray` class). If both methods are found, each `NSMutableArray` message sent to the collection proxy object results in some combination of `insertObject:in<Key>AtIndex:` and `removeObjectFrom<Key>AtIndex:` messages being sent to the original receiver of `mutableArrayValueForKey:`, unless the class of the receiver also implements an optional method whose name matches the pattern `replaceObjectIn<Key>AtIndex:withObject:`, in which case that replacement method will be used when appropriate for best performance.

2. Otherwise (no set of mutable array primitive methods is found), searches the class of the receiver for an accessor method whose name matches the pattern `set<Key>:`. If such a method is found, each `NSMutableArray` message sent to the collection proxy object results in a `set<Key>:` message being sent to the original receiver of `mutableArrayValueForKey:`.

3. Otherwise (no set of mutable array primitive methods or simple accessor method is found), if the receiver's `accessInstanceVariablesDirectly` class method returns `YES`, searches the class of the receiver for an instance variable whose name matches the pattern `_<key>` or `<key>`, in that order. If such an instance variable is found, each `NSMutableArray` message sent to the collection proxy object will be forwarded to the instance variable's value, which therefore must typically be an instance of `NSMutableArray` or a subclass of `NSMutableArray`.

4. Otherwise (no set of mutable array primitives, simple accessor method, or instance variable is found), raises an `NSUndefinedKeyException`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- mutableArrayValueForKeyPath:` (page 1269)

Declared In
`NSKeyValueCoding.h`

mutableArrayValueForKeyPath:

Returns a mutable array that provides read-write access to the ordered to-many relationship specified by a given key path.

```
- (NSMutableArray *)mutableArrayValueForKeyPath:(NSString *)keyPath
```

Parameters
keyPath

A key path, relative to the receiver, to an ordered to-many relationship.

Return Value
A mutable array that provides read-write access to the ordered to-many relationship specified by *keyPath*.

Discussion
See `mutableArrayValueForKey:` (page 1268) for additional details.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- mutableArrayValueForKey:` (page 1268)

Declared In
`NSKeyValueCoding.h`

mutableSetValueForKey:

Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key.

```
- (NSMutableSet *)mutableSetValueForKey:(NSString *)key
```

Parameters

key

The name of an unordered to-many relationship.

Return Value

A mutable set that provides read-write access to the unordered to-many relationship specified by *key*.

Discussion

Objects added to the mutable set become related to the receiver, and objects removed from the mutable set become unrelated. The default implementation recognizes the same simple accessor methods and set accessor methods as `valueForKey:` (page 1276), and follows the same direct instance variable access policies, but always returns a mutable collection proxy object instead of the immutable collection that `valueForKey:` would return.

This method also:

1. Searches the class of the receiver for a pair of methods whose names match the patterns `add<Key>Object:` and `remove<Key>Object:` (and therefore correspond to the two most primitive methods defined by the `NSMutableSet` class) and also `add<Key>:` and `remove<Key>:` (and therefore corresponding `NSMutableSet`'s `unionSet:` and `minusSet:`). If at least one addition method and at least one removal method are found each `NSMutableSet` message sent to the collection proxy object will result in some combination of `add<Key>Object:`, `remove<Key>Object:`, `add<Key>:`, and `remove<Key>:` messages being sent to the original receiver of `mutableSetValueForKey:`. If the receiver also implements an optional method whose name matches the pattern `intersect<Key>:` or `set<Key>:` that method will be used when appropriate for best performance.

2. Otherwise (no set of set mutation methods is found), searches the class of the receiver for an accessor method whose name matches the pattern `set<Key>:`. If such a method is found, each `NSMutableSet` message sent to the collection proxy object results in a `set<Key>:` message being sent to the original receiver of `mutableSetValueForKey:`.

3. Otherwise (no set of set mutation methods or simple accessor method is found), if the receiver's `accessInstanceVariablesDirectly` class method returns `YES`, searches the class of the receiver for an instance variable whose name matches the pattern `_<key>` or `<key>`, in that order. If such an instance variable is found, each `NSMutableSet` message sent to the collection proxy object will be forwarded to the instance variable's value, which therefore must typically be an instance of `NSMutableSet` or a subclass of `NSMutableSet`.

4. Otherwise (no set of mutable array primitives, simple accessor method, or instance variable is found), raises an `NSUndefinedKeyException`.

Note: The repetitive `set<Key>:` messages implied by Step 2's description are a potential performance problem. For better performance implement methods that fulfill the requirements for Step 1 in your KVC-compliant class.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- mutableArrayValueForKeyPath:` (page 1269)

Declared In
`NSKeyValueCoding.h`

mutableSetValueForKeyPath:

Returns a mutable set that provides read-write access to the unordered to-many relationship specified by a given key path.

```
- (NSMutableSet *)mutableSetValueForKeyPath:(NSString *)keyPath
```

Parameters
keyPath
A key path, relative to the receiver, to an unordered to-many relationship.

Return Value
A mutable set that provides read-write access to the unordered to-many relationship specified by *keyPath*.

Discussion
See `mutableSetValueForKey:` (page 1270) for additional details.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- mutableArrayValueForKey:` (page 1268)

Declared In
`NSKeyValueCoding.h`

setNilValueForKey:

Invoked by `setValue:forKey:` (page 1272) when it's given a `nil` value for a scalar value (such as an `int` or `float`).

```
- (void)setNilValueForKey:(NSString *)key
```

Parameters
key
The name of one of the receiver's properties.

Discussion
Subclasses can override this method to handle the request in some other way, such as by substituting `0` or a sentinel value for `nil` and invoking `setValue:forKey:` again or setting the variable directly. The default implementation raises an `NSInvalidArgumentException`.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyValueCoding.h`

setValue:forKey:
Sets the property of the receiver specified by a given key to a given value.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters
value
The value for the property identified by *key*.

key
The name of one of the receiver's properties.

Discussion
If *key* identifies a to-one relationship, relate the object specified by *value* to the receiver, unrelating the previously related object if there was one. Given a collection object and a *key* that identifies a to-many relationship, relate the objects contained in the collection to the receiver, unrelating previously related objects if there were any.

The default implementation of this method does the following:

1. Searches the class of the receiver for an accessor method whose name matches the pattern `-set<Key>:`. If such a method is found the type of its parameter is checked. If the parameter type is one of the data types supported by `NSNumber` or `NSValue` but the value is `nil`, `setNilValueForKey:` (page 1271) is invoked. If the value is not `nil` the appropriate `-<type>Value:` message is sent to the *value* and the result is used as the argument of an invocation of the accessor method. If the type of the accessor method's parameter is not an `NSNumber` or `NSValue` data type the accessor method is invoked with *value* as the argument.

2. Otherwise (no accessor method is found), if the receiver's `accessInstanceVariablesDirectly` class method returns `YES`, searches the class of the receiver for an instance variable whose name matches the pattern `_<key>`, `_is<Key>`, `<key>`, or `is<Key>`, in that order. If such an instance variable is found and its type is an `NSNumber` or `NSValue` data type, the value of the instance variable in the receiver is set using the same conversion as in step 1. If such an instance variable is found but its type is not an `NSNumber` or `NSValue` data type, the value is retained and the result is set in the instance variable, after first autoreleasing the instance variable's old value.

3. Otherwise (no accessor method or instance variable is found), invokes `setValue:forUndefinedKey:` (page 1274).

Compatibility notes:

- For backward binary compatibility with `takeValue:forKey:`'s behavior, a method whose name matches the pattern `-_set<Key>:` is also recognized in step 1.

  Note that KVC accessor methods whose names start with underscores are deprecated as of Mac OS X v10.3.
- For backward binary compatibility, `unableToSetNilForKey:` will be invoked instead of `setNilValueForKey:` in step 1, if the implementation of `unableToSetNilForKey:` in the receiver's class is not `NSObject`'s.
- The behavior described in step 2 is different from `takeValue:forKey:`'s, in which the instance variable search order is *<key>*, *_<key>*.
- For backward binary compatibility with the behavior of `takeValue:forKey:`, `handleTakeValue:forUnboundKey:` will be invoked instead of `valueForUndefinedKey:` in step 3 if the implementation of `handleTakeValue:forUnboundKey:` in the receiver's class is not `NSObject`'s.

Note: When the receiver is an instance of a Java class, member functions whose names match the pattern `set<Key>()` will be recognized in step 1.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyValueCoding.h`

setValue:forKeyPath:

Sets the value for the property identified by a given key path to a given value.

```
- (void)setValue:(id)value forKeyPath:(NSString *)keyPath
```

Parameters
value

The value for the property identified by *keyPath*.

keyPath

A key path of the form *relationship.property* (with one or more relationships): for example “department.name” or “department.manager.lastName.”

Discussion
The default implementation of this method gets the destination object for each relationship using `valueForKey:` (page 1276), and sends the final object a `setValue:forKey:` message.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- valueForKeyPath:` (page 1277)

Declared In
`NSKeyValueCoding.h`

setValue:forUndefinedKey:

Invoked by `setValue:forKey:` (page 1272) when it finds no property for a given key.

```
- (void)setValue:(id)value forUndefinedKey:(NSString *)key
```

Parameters

value

The value for the key identified by *key*.

key

A string that is not equal to the name of any of the receiver's properties.

Discussion

Subclasses can override this method to handle the request in some other way. The default implementation raises an `NSUndefinedKeyException`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- valueForUndefinedKey:` (page 1277)

Declared In

`NSKeyValueCoding.h`

setValuesForKeysWithDictionary:

Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties.

```
- (void)setValuesForKeysWithDictionary:(NSDictionary *)keyedValues
```

Parameters

keyedValues

A dictionary whose keys identify properties in the receiver. The values of the properties in the receiver are set to the corresponding values in the dictionary.

Discussion

The default implementation invokes `setValue:forKey:` (page 1272) for each key-value pair, substituting `nil` for `NSNull` values in *keyedValues*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- dictionaryWithValuesForKeys:` (page 1267)

Declared In

`NSKeyValueCoding.h`

validateValue:forKey:error:

Returns a Boolean value that indicates whether the value specified by a given pointer is valid for the property identified by a given key.

```
- (BOOL)validateValue:(id *)ioValue forKey:(NSString *)key error:(NSError **)outError
```

Parameters

ioValue

A pointer to a new value for the property identified by *key*. This method may modify or replace the value in order to make it valid.

key

The name of one of the receiver's properties. The key must specify an attribute or a to-one relationship.

outError

If validation is necessary and *ioValue* is not transformed into a valid value, upon return contains an `NSError` object that describes the reason that *ioValue* is not a valid value.

Return Value

YES if **ioValue* is a valid value for the property identified by *key*, or of the method is able to modify the value to **ioValue* to make it valid; otherwise `NO`.

Discussion

The default implementation of this method searches the class of the receiver for a validation method whose name matches the pattern `validate<Key>:error:`. If such a method is found it is invoked and the result is returned. If no such method is found, `YES` is returned.

The sender of the message is never given responsibility for releasing *ioValue* or *outError*.

See "Key-Value Validation" for more information.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- validateValue:forKeyPath:error:` (page 1275)

Declared In

`NSKeyValueCoding.h`

validateValue:forKeyPath:error:

Returns a Boolean value that indicates whether the value specified by a given pointer is valid for a given key path relative to the receiver.

```
- (BOOL)validateValue:(id *)ioValue forKeyPath:(NSString *)inKeyPath error:(NSError
    **)outError
```

Parameters

ioValue

A pointer to a new value for the property identified by *keyPath*. This method may modify or replace the value in order to make it valid.

key

The name of one of the receiver's properties. The key path must specify an attribute or a to-one relationship. The key path has the form *relationship.property* (with one or more relationships); for example "department.name" or "department.manager.lastName".

outError

If validation is necessary and *ioValue* is not transformed into a valid value, upon return contains an `NSError` object that describes the reason that *ioValue* is not a valid value.

Discussion

The default implementation gets the destination object for each relationship using `valueForKey:` (page 1276) and returns the result of a `validateValue:forKey:error:` message to the final object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- validateValue:forKey:error:` (page 1274)

Declared In

`NSKeyValueCoding.h`

valueForKey:

Returns the value for the property identified by a given key.

```
- (id)valueForKey:(NSString *)key
```

Parameters

key

The name of one of the receiver's properties.

Return Value

The value for the property identified by *key*.

Discussion

The default implementation works as follows:

1. Searches for a public accessor method based on *key*. For example, with a *key* of "lastName", `valueForKey:` looks for a method named `getLastName` or `lastName`.

2. If a public accessor method is not found and the class method `accessInstanceVariablesDirectly` (page 1267) returns `YES`, searches for a private accessor method based on *key* (a method preceded by an underbar). For example, with a *key* of "lastName", `valueForKey:` looks for a method named `_getLastName` or `_lastName`.

3. If an accessor method is not found `valueForKey:` searches for an instance variable based on *key* and returns its value directly. For the *key* "lastName", this would be `_lastName` or `lastName`.

4. If neither an accessor method nor an instance variable is found, the default implementation invokes `valueForUndefinedKey:` (page 1277).

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyValueCoding.h`

valueForKeyPath:

Returns the value for the derived property identified by a given key path.

```
- (id)valueForKeyPath:(NSString *)keyPath
```

Parameters

keyPath

A key path of the form *relationship.property* (with one or more relationships); for example “department.name” or “department.manager.lastName”.

Return Value

The value for the derived property identified by *keyPath*.

Discussion

The default implementation gets the destination object for each relationship using `valueForKey:` (page 1276) and returns the result of a `valueForKey:` message to the final object.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setValue:forKeyPath:` (page 1273)

Declared In

`NSKeyValueCoding.h`

valueForUndefinedKey:

Invoked by `valueForKey:` (page 1276) when it finds no property corresponding to a given key.

```
- (id)valueForUndefinedKey:(NSString *)key
```

Parameters

key

A string that is not equal to the name of any of the receiver's properties.

Discussion

Subclasses can override this method to return an alternate value for undefined keys. The default implementation raises an `NSUndefinedKeyException`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`- setValue:forUndefinedKey:` (page 1274)

Declared In

`NSKeyValueCoding.h`

Constants

Key Value Coding Exception Names

This constant defines the name of an exception raised when a key value coding operation fails.

```
extern NSString *NSUndefinedKeyException;
```

Constants

`NSUndefinedKeyException`

Raised when a key value coding operation fails. *`userInfo`* keys are described in "NSUndefinedKeyException userInfo Keys" (page 1278)

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

Declared In

`NSKeyValueCoding.h`

NSUndefinedKeyException userInfo Keys

These constants are keys into an `NSUndefinedKeyException` *`userInfo`* dictionary

```
extern NSString *NSTargetObjectUserInfoKey;
extern NSString *NSUnknownUserInfoKey;
```

Constants

`NSTargetObjectUserInfoKey`

The object on which the key value coding operation failed.

`NSUnknownUserInfoKey`

The key for which the key value coding operation failed.

Discussion

For additional information see "Key Value Coding Exception Names" (page 1278).

Declared In

`NSKeyValueCoding.h`

Array operators

These constants define the available array operators. See Set and Array Operators for more information.

```
NSString *const NSAverageKeyValueOperator;
NSString *const NSCountKeyValueOperator;
NSString *const NSDistinctUnionOfArraysKeyValueOperator;
NSString *const NSDistinctUnionOfObjectsKeyValueOperator;
NSString *const NSDistinctUnionOfSetsKeyValueOperator;
NSString *const NSMaximumKeyValueOperator;
NSString *const NSMinimumKeyValueOperator;
NSString *const NSSumKeyValueOperator;
NSString *const NSUnionOfArraysKeyValueOperator;
NSString *const NSUnionOfObjectsKeyValueOperator;
NSString *const NSUnionOfSetsKeyValueOperator;
```

Constants

`NSAverageKeyValueOperator`

The `@avg` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSCountKeyValueOperator`

The `@count` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSDistinctUnionOfArraysKeyValueOperator`

The `@distinctUnionOfArrays` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSDistinctUnionOfObjectsKeyValueOperator`

The `@distinctUnionOfObjects` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSDistinctUnionOfSetsKeyValueOperator`

The `@distinctUnionOfSets` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSMaximumKeyValueOperator`

The `@max` array operator.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueCoding.h`

`NSMinimumKeyValueOperator`
The `@min` array operator.
Available in Mac OS X v10.4 and later.
Available in iPhone OS 2.0 and later.
Declared in `NSKeyValueCoding.h`

`NSSumKeyValueOperator`
The `@sum` array operator.
Available in Mac OS X v10.4 and later.
Available in iPhone OS 2.0 and later.
Declared in `NSKeyValueCoding.h`

`NSUnionOfArraysKeyValueOperator`
The `@unionOfArrays` array operator.
Available in Mac OS X v10.4 and later.
Available in iPhone OS 2.0 and later.
Declared in `NSKeyValueCoding.h`

`NSUnionOfObjectsKeyValueOperator`
The `@unionOfObjects` array operator.
Available in Mac OS X v10.4 and later.
Available in iPhone OS 2.0 and later.
Declared in `NSKeyValueCoding.h`

`NSUnionOfSetsKeyValueOperator`
The `@unionOfSets` array operator.
Available in Mac OS X v10.4 and later.
Available in iPhone OS 2.0 and later.
Declared in `NSKeyValueCoding.h`

Availability
Available in Mac OS X version 10.4 and later.

Declared In
`NSKeyValueCoding.h`

NSKeyValueObserving Protocol Reference

(informal protocol)

Framework	/System/Library/Frameworks/Foundation.framework
Declared in:	Foundation/NSKeyValueObserving.h
Availability:	Available in Mac OS X v10.3 and later.
Companion guide:	Key-Value Observing Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSKeyValueObserving` (KVO) informal protocol defines a mechanism that allows objects to be notified of changes to the specified properties of other objects.

You can observe any object properties including simple attributes, to-one relationships, and to-many relationships. Observers of to-many relationships are informed of the type of change made — as well as which objects are involved in the change.

`NSObject` provides an implementation of the `NSKeyValueObserving` protocol that provides an automatic observing capability for all objects. You can further refine notifications by disabling automatic observer notifications and implementing manual notifications using the methods in this protocol.

Note: Key-value observing is not available for Java applications.

Tasks

Change Notification

- `observeValueForKeyPath:ofObject:change:context:` (page 1287)
  This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

Registering for Observation

- `addObserver:forKeyPath:options:context:` (page 1284)
  Registers *anObserver* to receive KVO notifications for the specified key-path relative to the receiver.
- `removeObserver:forKeyPath:` (page 1288)
  Stops a given object from receiving change notifications for the property specified by a given key-path relative to the receiver.

Notifying Observers of Changes

- `willChangeValueForKey:` (page 1289)
  Invoked to inform the receiver that the value of a given property is about to change.
- `didChangeValueForKey:` (page 1285)
  Invoked to inform the receiver that the value of a given property has changed.
- `willChange:valuesAtIndexes:forKey:` (page 1289)
  Invoked to inform the receiver that the specified change is about to be executed at given indexes for a specified ordered to-many relationship.
- `didChange:valuesAtIndexes:forKey:` (page 1285)
  Invoked to inform the receiver that the specified change has occurred on the indexes for a specified ordered to-many relationship.
- `willChangeValueForKey:withSetMutation:usingObjects:` (page 1290)
  Invoked to inform the receiver that the specified change is about to be made to a specified unordered to-many relationship.
- `didChangeValueForKey:withSetMutation:usingObjects:` (page 1286)
  Invoked to inform the receiver that the specified change was made to a specified unordered to-many relationship.

Observing Customization

`+ automaticallyNotifiesObserversForKey:` (page 1283)
Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key.

`+ keyPathsForValuesAffectingValueForKey:` (page 1283)
Returns a set of key paths for properties whose values affect the value of the specified key.

`- setObservationInfo:` (page 1288)
Sets the observation info for the receiver.

`- observationInfo` (page 1286)
Returns a pointer that identifies information about all of the observers that are registered with the receiver.

Class Methods

automaticallyNotifiesObserversForKey:

Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key.

```
+ (BOOL)automaticallyNotifiesObserversForKey:(NSString *)key
```

Return Value

`YES` if the key-value observing machinery should automatically invoke `willChangeValueForKey:` (page 1289)/`didChangeValueForKey:` (page 1285) and `willChange:valuesAtIndexes:forKey:` (page 1289)/`didChange:valuesAtIndexes:forKey:` (page 1285) whenever instances of the class receive key-value coding messages for the *key*, or mutating key-value-coding-compliant methods for the *key* are invoked; otherwise `NO`.

Discussion

The default implementation returns `YES`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSKeyValueObserving.h`

keyPathsForValuesAffectingValueForKey:

Returns a set of key paths for properties whose values affect the value of the specified key.

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key
```

Parameters

key
The key whose value is affected by the key paths.

Return Value

Discussion
When an observer for the key is registered with an instance of the receiving class, key-value observing itself automatically observes all of the key paths for the same instance, and sends change notifications for the key to the observer when the value for any of those key paths changes.

The default implementation of this method searches the receiving class for a method whose name matches the pattern `+keyPathsForValuesAffecting<Key>`, and returns the result of invoking that method if it is found. Any such method must return an NSSet. If no such method is found, an NSSet that is computed from information provided by previous invocations of the now-deprecated `setKeys:triggerChangeNotificationsForDependentKey:` method is returned, for backward binary compatibility.

You can override this method when the getter method of one of your properties computes a value to return using the values of other properties, including those that are located by key paths. Your override should typically invoke super and return a set that includes any members in the set that result from doing that (so as not to interfere with overrides of this method in superclasses).

Note: You must not override this method when you add a computed property to an existing class using a category, overriding methods in categories is unsupported. In that case, implement a matching `+keyPathsForValuesAffecting<Key>` to take advantage of this mechanism.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyValueObserving.h`

Instance Methods

addObserver:forKeyPath:options:context:

Registers *anObserver* to receive KVO notifications for the specified key-path relative to the receiver.

```
- (void)addObserver:(NSObject *)anObserver
    forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options
    context:(void *)context
```

Parameters

anObserver
: The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:` (page 1287).

keyPath
: The key path, relative to the receiver, of the property to observe. This value must not be `nil`.

options
: A combination of the `NSKeyValueObservingOptions` values that specifies what is included in observation notifications. For possible values, see NSKeyValueObservingOptions (page 1291).

context

Arbitrary data that is passed to *anObserver* in observeValueForKeyPath:ofObject:change:context: (page 1287).

Discussion

Neither the receiver, nor *anObserver*, are retained.

Availability

Available in iPhone OS 2.0 and later.

See Also

- removeObserver:forKeyPath: (page 1288)

Declared In

NSKeyValueObserving.h

didChange:valuesAtIndexes:forKey:

Invoked to inform the receiver that the specified change has occurred on the indexes for a specified ordered to-many relationship.

```
- (void)didChange:(NSKeyValueChange)change
    valuesAtIndexes:(NSIndexSet *)indexes
    forKey:(NSString *)key
```

Parameters

change

The type of change that was made.

indexes

The indexes of the to-many relationship that were affected by the change.

key

The name of a property that is an ordered to-many relationship.

Discussion

You should invoke this method when implementing key-value-observing compliance manually.

Availability

Available in iPhone OS 2.0 and later.

See Also

- willChange:valuesAtIndexes:forKey: (page 1289)
- didChangeValueForKey: (page 1285)

Declared In

NSKeyValueObserving.h

didChangeValueForKey:

Invoked to inform the receiver that the value of a given property has changed.

```
- (void)didChangeValueForKey:(NSString *)key
```

Parameters

key

The name of the property that changed.

Discussion

You should invoke this method when implementing key-value observer compliance manually.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `willChangeValueForKey:` (page 1289)
- `didChange:valuesAtIndexes:forKey:` (page 1285)

Declared In

`NSKeyValueObserving.h`

didChangeValueForKey:withSetMutation:usingObjects:

Invoked to inform the receiver that the specified change was made to a specified unordered to-many relationship.

```
- (void)didChangeValueForKey:(NSString *)key
    withSetMutation:(NSKeyValueSetMutationKind)mutationKind
    usingObjects:(NSSet *)objects
```

Parameters

key

The name of a property that is an unordered to-many relationship

mutationKind

The type of change that was made.

objects

The objects that were involved in the change (see NSKeyValueSetMutationKind (page 1294)).

Discussion

You invoke this method when implementing key-value observer compliance manually.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `willChangeValueForKey:withSetMutation:usingObjects:` (page 1290)

Declared In

`NSKeyValueObserving.h`

observationInfo

Returns a pointer that identifies information about all of the observers that are registered with the receiver.

```
- (void *)observationInfo
```

Return Value
A pointer that identifies information about all of the observers that are registered with the receiver, the options that were used at registration-time, and so on.

Discussion
The default implementation of this method retrieves the information from a global dictionary keyed by the receiver's pointers.

For improved performance, this method and `setObservationInfo:` can be overridden to store the opaque data pointer in an instance variable. Overrides of this method must not attempt to send Objective-C messages to the stored data, including `retain` and `release`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- setObservationInfo:` (page 1288)

Declared In
`NSKeyValueObserving.h`

observeValueForKeyPath:ofObject:change:context:

This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
    ofObject:(id)object
    change:(NSDictionary *)change
    context:(void *)context
```

Parameters
keyPath
: The key path, relative to *object*, to the value that has changed.

object
: The source object of the key path *keyPath*.

change
: A dictionary that describes the changes that have been made to the value of the property at the key path *keyPath* relative to *object*. Entries are described in "Keys used by the change dictionary" (page 1293).

context
: The value that was provided when the receiver was registered to receive key-value observation notifications.

Discussion
The receiver must be registered as an observer for the specified *keyPath* and *object*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSKeyValueObserving.h`

removeObserver:forKeyPath:

Stops a given object from receiving change notifications for the property specified by a given key-path relative to the receiver.

```
- (void)removeObserver:(NSObject *)anObserver
    forKeyPath:(NSString *)keyPath
```

Parameters

anObserver

The object to remove as an observer.

keyPath

A key-path, relative to the receiver, for which *anObserver* is registered to receive KVO change notifications.

Availability

Available in iPhone OS 2.0 and later.

See Also

- addObserver:forKeyPath:options:context: (page 1284)

Declared In

NSKeyValueObserving.h

setObservationInfo:

Sets the observation info for the receiver.

```
- (void)setObservationInfo:(void *)observationInfo
```

Parameters

observationInfo

The observation info for the receiver.

Discussion

The *observationInfo* is a pointer that identifies information about all of the observers that are registered with the receiver. The default implementation of this method stores *observationInfo* in a global dictionary keyed by the receiver's pointers.

For improved performance, this method and observationInfo can be overridden to store the opaque data pointer in an instance variable. Classes that override this method must not attempt to send Objective-C messages to *observationInfo*, including retain and release.

Availability

Available in iPhone OS 2.0 and later.

See Also

- observationInfo (page 1286)

Declared In

NSKeyValueObserving.h

willChange:valuesAtIndexes:forKey:

Invoked to inform the receiver that the specified change is about to be executed at given indexes for a specified ordered to-many relationship.

```
- (void)willChange:(NSKeyValueChange)change
    valuesAtIndexes:(NSIndexSet *)indexes
    forKey:(NSString *)key
```

Parameters

change

The type of change that is about to be made.

indexes

The indexes of the to-many relationship that will be affected by the change.

key

The name of a property that is an ordered to-many relationship.

Discussion

You should invoke this method when implementing key-value-observing compliance manually.

Important: After the values have been changed, a corresponding `didChange:valuesAtIndexes:forKey:` (page 1285) must be invoked with the same parameters.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `didChange:valuesAtIndexes:forKey:` (page 1285)
- `willChangeValueForKey:` (page 1289)

Declared In

`NSKeyValueObserving.h`

willChangeValueForKey:

Invoked to inform the receiver that the value of a given property is about to change.

```
- (void)willChangeValueForKey:(NSString *)key
```

Parameters

key

The name of the property that will change.

Discussion

You should invoke this method when implementing key-value observer compliance manually.

The change type of this method is `NSKeyValueChangeSetting`.

Important: After the values have been changed, a corresponding `didChangeValueForKey:` (page 1285) must be invoked with the same parameter.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `didChangeValueForKey:` (page 1285)
- `willChange:valuesAtIndexes:forKey:` (page 1289)

Declared In
`NSKeyValueObserving.h`

willChangeValueForKey:withSetMutation:usingObjects:
Invoked to inform the receiver that the specified change is about to be made to a specified unordered to-many relationship.

```
- (void)willChangeValueForKey:(NSString *)key
    withSetMutation:(NSKeyValueSetMutationKind)mutationKind
    usingObjects:(NSSet *)objects
```

Parameters
key
The name of a property that is an unordered to-many relationship

mutationKind
The type of change that will be made.

objects
The objects that are involved in the change (see NSKeyValueSetMutationKind (page 1294)).

Discussion
You invoke this method when implementing key-value observer compliance manually.

Important: After the values have been changed, a corresponding `didChangeValueForKey:withSetMutation:usingObjects:` (page 1286) must be invoked with the same parameters.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `didChangeValueForKey:withSetMutation:usingObjects:` (page 1286)

Declared In
`NSKeyValueObserving.h`

Constants

NSKeyValueChange

These constants are returned as the value for a `NSKeyValueChangeKindKey` key in the change dictionary passed to `observeValueForKeyPath:ofObject:change:context:` (page 1287) indicating the type of change made:

```
enum {
    NSKeyValueChangeSetting = 1,
    NSKeyValueChangeInsertion = 2,
    NSKeyValueChangeRemoval = 3,
    NSKeyValueChangeReplacement = 4
};
typedef NSUInteger NSKeyValueChange;
```

Constants

`NSKeyValueChangeSetting`

Indicates that the value of the observed key path was set to a new value. This change can occur when observing an attribute of an object, as well as properties that specify to-one and to-many relationships.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeInsertion`

Indicates that an object has been inserted into the to-many relationship that is being observed.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeRemoval`

Indicates that an object has been removed from the to-many relationship that is being observed.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeReplacement`

Indicates that an object has been replaced in the to-many relationship that is being observed.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

Declared In

`NSKeyValueObserving.h`

NSKeyValueObservingOptions

These constants are passed to `addObserver:forKeyPath:options:context:` (page 1284) and determine the values that are returned as part of the change dictionary passed to an `observeValueForKeyPath:ofObject:change:context:` (page 1287). You can pass `0` if you require no change dictionary values.

```
enum {
    NSKeyValueObservingOptionNew = 0x01,
    NSKeyValueObservingOptionOld = 0x02,
    NSKeyValueObservingOptionInitial = 0x04,
    NSKeyValueObservingOptionPrior = 0x08
};
typedef NSUInteger NSKeyValueObservingOptions;
```

Constants

`NSKeyValueObservingOptionNew`

Indicates that the change dictionary should provide the new attribute value, if applicable.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueObservingOptionOld`

Indicates that the change dictionary should contain the old attribute value, if applicable.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueObservingOptionInitial`

If specified, a notification should be sent to the observer immediately, before the observer registration method even returns. The change dictionary in the notification will always contain an `NSKeyValueChangeNewKey` entry if `NSKeyValueObservingOptionNew` is also specified but will never contain an `NSKeyValueChangeOldKey` entry. (In an initial notification the current value of the observed property may be old, but it's new to the observer.) You can use this option instead of explicitly invoking, at the same time, code that is also invoked by the observer's `observeValueForKeyPath:ofObject:change:context:` method. When this option is used with`addObserver:forKeyPath:options:context:` a notification will be sent for each indexed object to which the observer is being added.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueObservingOptionPrior`

Whether separate notifications should be sent to the observer before and after each change, instead of a single notification after the change. The change dictionary in a notification sent before a change always contains an `NSKeyValueChangeNotificationIsPriorKey` entry whose value is `[NSNumber numberWithBool:YES]`, but never contains an `NSKeyValueChangeNewKey` entry. When this option is specified the change dictionary in a notification sent after a change contains the same entries that it would contain if this option were not specified. You can use this option when the observer's own key-value observing-compliance requires it to invoke one of the `-willChange...` methods for one of its own properties, and the value of that property depends on the value of the observed object's property. (In that situation it's too late to easily invoke `-willChange...` properly in response to receiving an `observeValueForKeyPath:ofObject:change:context:` message after the change.)

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

Declared In

`NSKeyValueObserving.h`

Keys used by the change dictionary

These constants are used as keys in the change dictionary passed to `observeValueForKeyPath:ofObject:change:context:` (page 1287).

```
NSString *const NSKeyValueChangeKindKey;
NSString *const NSKeyValueChangeNewKey;
NSString *const NSKeyValueChangeOldKey;
NSString *const NSKeyValueChangeIndexesKey;
```

Constants

`NSKeyValueChangeKindKey`

An `NSNumber` object that contains a value corresponding to one of the `NSKeyValueChangeKindKey` enumerations, indicating what sort of change has occurred.

A value of `NSKeyValueChangeSetting` indicates that the observed object has received a `setValue:forKey:` message, or that the key-value-coding-compliant set method for the key has been invoked, or that `willChangeValueForKey:` (page 1289)/`didChangeValueForKey:` (page 1285) has otherwise been invoked.

A value of `NSKeyValueChangeInsertion`, `NSKeyValueChangeRemoval`, or `NSKeyValueChangeReplacement` indicates that mutating messages have been sent to the array returned by a `mutableArrayValueForKey:` message sent to the object, or that one of the key-value-coding-compliant array mutation methods for the key has been invoked, or that `willChange:valuesAtIndexes:forKey:` (page 1289)/`didChange:valuesAtIndexes:forKey:` (page 1285) has otherwise been invoked.

You can use `NSNumber`'s `intValue` (page 713) method to retrieve the integer value of the change kind.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeNewKey`

If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeSetting`, and `NSKeyValueObservingOptionNew` was specified when the observer was registered, the value of this key is the new value for the attribute.

For `NSKeyValueChangeInsertion` or `NSKeyValueChangeReplacement`, if `NSKeyValueObservingOptionNew` was specified when the observer was registered, the value for this key is an `NSArray` instance that contains the objects that have been inserted or replaced other objects, respectively.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeOldKey`

If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeSetting`, and `NSKeyValueObservingOptionOld` was specified when the observer was registered, the value of this key is the value before the attribute was changed.

For `NSKeyValueChangeRemoval` or `NSKeyValueChangeReplacement`, if `NSKeyValueObservingOptionOld` was specified when the observer was registered, the value is an `NSArray` instance that contains the objects that have been removed or have been replaced by other objects, respectively.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueChangeIndexesKey`
If the value of the `NSKeyValueChangeKindKey` entry is `NSKeyValueChangeInsertion`, `NSKeyValueChangeRemoval`, or `NSKeyValueChangeReplacement`, the value of this key is an `NSIndexSet` object that contains the indexes of the inserted, removed, or replaced objects.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

Declared In
`NSKeyValueObserving.h`

NSKeyValueSetMutationKind

These constants are specified as the parameter to the methods `willChangeValueForKey:withSetMutation:usingObjects:` (page 1290) and `didChangeValueForKey:withSetMutation:usingObjects:` (page 1286).

```
enum {
    NSKeyValueUnionSetMutation = 1,
    NSKeyValueMinusSetMutation = 2,
    NSKeyValueIntersectSetMutation = 3,
    NSKeyValueSetSetMutation = 4

};
typedef NSUInteger NSKeyValueSetMutationKind;
```

Constants
`NSKeyValueUnionSetMutation`
Indicates that objects in the specified set are being added to the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeInsertion`.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueMinusSetMutation`
Indicates that the objects in the specified set are being removed from the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeRemoval`.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueIntersectSetMutation`
Indicates that the objects not in the specified set are being removed from the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeRemoval`.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

`NSKeyValueSetSetMutation`

Indicates that set of objects are replacing the existing objects in the receiver. This mutation kind results in a `NSkeyValueChangeKindKey` value of `NSKeyValueChangeReplacement`.

Available in Mac OS X v10.4 and later.

Available in iPhone OS 2.0 and later.

Declared in `NSKeyValueObserving.h`

Declared In
`NSKeyValueObserving.h`

NSLocking Protocol Reference

Adopted by:	NSConditionLock NSLock NSRecursiveLock
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSLock.h
Companion guide:	Threading Programming Guide

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSLocking` protocol declares the elementary methods adopted by classes that define lock objects. A lock object is used to coordinate the actions of multiple threads of execution within a single application. By using a lock object, an application can protect critical sections of code from being executed simultaneously by separate threads, thus protecting shared data and other shared resources from corruption.

Tasks

Working with Locks

- `lock` (page 1298)
  Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired.
- `unlock` (page 1298)
  Relinquishes a previously acquired lock.

Instance Methods

lock

Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired.

```
- (void)lock
```

Discussion
An application protects a critical section of code by requiring a thread to acquire a lock before executing the code. Once the critical section is past, the thread relinquishes the lock by invoking `unlock` (page 1298).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLock.h`

unlock

Relinquishes a previously acquired lock.

```
- (void)unlock
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSLock.h`

NSMutableCopying Protocol Reference

Adopted by:	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSObject.h
Companion guide:	Memory Management Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSMutableCopying` protocol declares a method for providing mutable copies of an object. Only classes that define an "immutable vs. mutable" distinction should adopt this protocol. Classes that don't define such a distinction should adopt `NSCopying` instead.

`NSMutableCopying` declares one method, `mutableCopyWithZone:` (page 1300), but mutable copying is commonly invoked with the convenience method `mutableCopy`. The `mutableCopy` method is defined for all NSObjects and simply invokes `mutableCopyWithZone:` (page 1300) with the default zone.

If a subclass inherits NSMutableCopying from its superclass and declares additional instance variables, the subclass has to override `mutableCopyWithZone:` (page 1300) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

- mutableCopyWithZone: (page 1300)
  Returns a new instance that's a mutable copy of the receiver.

Instance Methods

mutableCopyWithZone:

Returns a new instance that's a mutable copy of the receiver.

```
- (id)mutableCopyWithZone:(NSZone *)zone
```

Parameters

zone

The zone from which memory is allocated for the new instance. If *zone* is NULL, the new instance is allocated from the default zone, which is returned by NSDefaultMallocZone (page 1357).

Discussion

The returned object is implicitly retained by the sender, which is responsible for releasing it. The copy returned is mutable whether the original is mutable or not.

Availability

Available in iPhone OS 2.0 and later.

See Also

- copyWithZone: (page 1250) (NSCopying protocol)
- mutableCopy (page 806) (NSObject class)

Declared In

NSObject.h

NSObject Protocol Reference

Adopted by:	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSObject.h
Companion guides:	Cocoa Fundamentals Guide Memory Management Programming Guide for Cocoa

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSObject` protocol groups methods that are fundamental to all Objective-C objects.

If an object conforms to this protocol, it can be considered a first-class object. Such an object can be asked about its:

- Class, and the place of its class in the inheritance hierarchy
- Conformance to protocols
- Ability to respond to a particular message

In addition, objects that conform to this protocol—with its retain (page 1312), release (page 1310), and autorelease (page 1303) methods—can also integrate with the object management and deallocation scheme defined in Foundation (for more information see, for example, *Memory Management Programming Guide for Cocoa*). Thus, an object that conforms to the `NSObject` protocol can be managed by container objects like those defined by `NSArray` and `NSDictionary`.

The Cocoa root class, `NSObject`, adopts this protocol, so all objects inheriting from `NSObject` have the features described by this protocol.

Tasks

Identifying Classes

- `class` (page 1304)
  Returns the class object for the receiver's class.
- `superclass` (page 1313)
  Returns the class object for the receiver's superclass.

Identifying and Comparing Objects

- `isEqual:` (page 1306)
  Returns a Boolean value that indicates whether the receiver and a given object are equal.
- `hash` (page 1305)
  Returns an integer that can be used as a table address in a hash table structure.
- `self` (page 1313)
  Returns the receiver.

Managing Reference Counts

- `retain` (page 1312)
  Increments the receiver's reference count.
- `release` (page 1310)
  Decrements the receiver's reference count.
- `autorelease` (page 1303)
  Adds the receiver to the current autorelease pool.
- `retainCount` (page 1312)
  Returns the receiver's reference count.

Testing Object Inheritance, Behavior, and Conformance

- `isKindOfClass:` (page 1306)
  Returns a Boolean value that indicates whether the receiver is an instance of given class or an instance of any class that inherits from that class.

- `isMemberOfClass:` (page 1307)
  Returns a Boolean value that indicates whether the receiver is an instance of a given class.
- `respondsToSelector:` (page 1311)
  Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message.
- `conformsToProtocol:` (page 1304)
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Describing Objects

- `description` (page 1305)
  Returns a string that describes the contents of the receiver.

Sending Messages

- `performSelector:` (page 1308)
  Sends a specified message to the receiver and returns the result of the message.
- `performSelector:withObject:` (page 1309)
  Sends a message to the receiver with an object as the argument.
- `performSelector:withObject:withObject:` (page 1309)
  Sends a message to the receiver with two objects as as arguments.

Determining Allocation Zones

- `zone` (page 1314)
  Returns a pointer to the zone from which the receiver was allocated.

Identifying Proxies

- `isProxy` (page 1308)
  Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`.

Instance Methods

autorelease

Adds the receiver to the current autorelease pool.

```
- (id)autorelease
```

Return Value
`self`.

Discussion
You add an object to an autorelease pool so it will receive a `release` message—and thus might be deallocated—when the pool is destroyed. For more information on the autorelease mechanism, see *Memory Management Programming Guide for Cocoa*.

Special Considerations
If garbage collection is enabled, this method is a no-op.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `retain` (page 1312)
- `retainCount` (page 1312)

Declared In
`NSObject.h`

class
Returns the class object for the receiver's class.

```
- (Class)class
```

Return Value
The class object for the receiver's class.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ class` (page 785) (`NSObject` class)

Declared In
`NSObject.h`

conformsToProtocol:
Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

```
- (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Parameters
aProtocol

A protocol object that represents a particular protocol.

Return Value
`YES` if the receiver conforms to *aProtocol*, otherwise `NO`.

Discussion
This method works identically to the `conformsToProtocol:` (page 787) class method declared in `NSObject`. It's provided as a convenience so that you don't need to get the class object to find out whether an instance can respond to a given set of messages.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

description

Returns a string that describes the contents of the receiver.

```
- (NSString *)description
```

Return Value
A string that describes the contents of the receiver.

Discussion
The debugger's print-object command indirectly invokes this method to produce a textual description of an object.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObject.h

hash

Returns an integer that can be used as a table address in a hash table structure.

```
- (NSUInteger)hash
```

Return Value
An integer that can be used as a table address in a hash table structure.

Discussion
If two objects are equal (as determined by the `isEqual:` (page 1306) method), they must have the same hash value. This last point is particularly important if you define `hash` in a subclass and intend to put instances of that subclass into a collection.

If a mutable object is added to a collection that uses hash values to determine the object's position in the collection, the value returned by the `hash` method of the object must not change while the object is in the collection. Therefore, either the `hash` method must not rely on any of the object's internal state information or you must make sure the object's internal state information does not change while the object is in the collection. Thus, for example, a mutable dictionary can be put in a hash table but you must not change it while it is in there. (Note that it can be difficult to know whether or not a given object is in a collection.)

Availability
Available in iPhone OS 2.0 and later.

See Also
- `isEqual:` (page 1306)

Declared In
NSObject.h

isEqual:

Returns a Boolean value that indicates whether the receiver and a given object are equal.

```
- (BOOL)isEqual:(id)anObject
```

Parameters

anObject
The object to be compared to the receiver.

Return Value
YES if the receiver and *anObject* are equal, otherwise NO.

Discussion
This method defines what it means for instances to be equal. For example, a container object might define two containers as equal if their corresponding objects all respond YES to an isEqual: request. See the NSData, NSDictionary, NSArray, and NSString class specifications for examples of the use of this method.

If two objects are equal, they must have the same hash value. This last point is particularly important if you define isEqual: in a subclass and intend to put instances of that subclass into a collection. Make sure you also define hash (page 1305) in your subclass.

Availability
Available in iPhone OS 2.0 and later.

See Also
- hash (page 1305)

Declared In
NSObject.h

isKindOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of given class or an instance of any class that inherits from that class.

```
- (BOOL)isKindOfClass:(Class)aClass
```

Parameters

aClass
A class object representing the Objective-C class to be tested.

Return Value
YES if the receiver is an instance of *aClass* or an instance of any class that inherits from *aClass*, otherwise NO.

Discussion
For example, in this code, isKindOfClass: would return YES because, in Foundation, the NSArchiver class inherits from NSCoder:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ( [anArchiver isKindOfClass:[NSCoder class]] )
    ...
```

Be careful when using this method on objects represented by a class cluster. Because of the nature of class clusters, the object you get back may not always be the type you expected. If you call a method that returns a class cluster, the exact type returned by the method is the best indicator of what you can do with that object. For example, if a method returns a pointer to an `NSArray` object, you should not use this method to see if the array is mutable, as shown in the following code:

```
// DO NOT DO THIS!
if ([myArray isKindOfClass:[NSMutableArray class]])
{
    // Modify the object
}
```

If you use such constructs in your code, you might think it is alright to modify an object that in reality should not be modified. Doing so might then create problems for other code that expected the object to remain unchanged.

If the receiver is a class object, this method returns `YES` if *`aClass`* is a Class object of the same type, `NO` otherwise.

Availability
Available in iPhone OS 2.0 and later.

See Also
`- isMemberOfClass:` (page 1307)

Declared In
`NSObject.h`

isMemberOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of a given class.

```
- (BOOL)isMemberOfClass:(Class)aClass
```

Parameters
`aClass`
A class object representing the Objective-C class to be tested.

Return Value
`YES` if the receiver is an instance of *`aClass`*, otherwise `NO`.

Discussion
For example, in this code, `isMemberOfClass:` would return `NO`:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ([anArchiver isMemberOfClass:[NSCoder class]])
    ...
```

Class objects may be compiler-created objects but they still support the concept of membership. Thus, you can use this method to verify that the receiver is a specific Class object.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `isKindOfClass:` (page 1306)

Declared In
`NSObject.h`

isProxy

Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`.

```
- (BOOL)isProxy
```

Return Value
`NO` if the receiver really descends from `NSObject`, otherwise `YES`.

Discussion
This method is necessary because sending `isKindOfClass:` (page 1306) or `isMemberOfClass:` (page 1307) to an `NSProxy` object will test the object the proxy stands in for, not the proxy itself. Use this method to test if the receiver is a proxy (or a member of some other root class).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSObject.h`

performSelector:

Sends a specified message to the receiver and returns the result of the message.

```
- (id)performSelector:(SEL)aSelector
```

Parameters

aSelector
A selector identifying the message to send. If *aSelector* is `NULL`, an `NSInvalidArgumentException` is raised.

Return Value
An object that is the result of the message.

Discussion
The `performSelector:` method is equivalent to sending an *aSelector* message directly to the receiver. For example, all three of the following messages do the same thing:

```
id myClone = [anObject copy];
id myClone = [anObject performSelector:@selector(copy)];
id myClone = [anObject performSelector:sel_getUid("copy")];
```

However, the `performSelector:` method allows you to send messages that aren't determined until runtime. A variable selector can be passed as the argument:

```
SEL myMethod = findTheAppropriateSelectorForTheCurrentSituation();
[anObject performSelector:myMethod];
```

The *aSelector* argument should identify a method that takes no arguments. For methods that return anything other than an object, use NSInvocation.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `performSelector:withObject:` (page 1309)
- `performSelector:withObject:withObject:` (page 1309)

Declared In
`NSObject.h`

performSelector:withObject:

Sends a message to the receiver with an object as the argument.

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters
aSelector
: A selector identifying the message to send. If *aSelector* is `NULL`, an `NSInvalidArgumentException` is raised.

anObject
: An object that is the sole argument of the message.

Return Value
An object that is the result of the message.

Discussion
This method is the same as `performSelector:` (page 1308) except that you can supply an argument for *aSelector*. *aSelector* should identify a method that takes a single argument of type `id`. For methods with other argument types and return values, use NSInvocation.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `performSelector:withObject:withObject:` (page 1309)
- `methodForSelector:` (page 805) (`NSObject` class)

Declared In
`NSObject.h`

performSelector:withObject:withObject:

Sends a message to the receiver with two objects as as arguments.

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
    withObject:(id)anotherObject
```

Parameters

aSelector
A selector identifying the message to send. If *aSelector* is NULL, an NSInvalidArgumentException is raised.

anObject
An object that is the first argument of the message.

anotherObject
An object that is the second argument of the message

Return Value

An object that is the result of the message.

Discussion

This method is the same as performSelector: (page 1308) except that you can supply two arguments for *aSelector*. *aSelector* should identify a method that can take two arguments of type id. For methods with other argument types and return values, use NSInvocation.

Availability

Available in iPhone OS 2.0 and later.

See Also

- performSelector:withObject: (page 1309)
- methodForSelector: (page 805) (NSObject class)

Declared In

NSObject.h

release

Decrements the receiver's reference count.

```
- (oneway void)release
```

Discussion

The receiver is sent a dealloc (page 799) message when its reference count reaches 0.

You would only implement this method to define your own reference-counting scheme. Such implementations should not invoke the inherited method; that is, they should not include a release message to super.

For more information on the reference counting mechanism, see *Memory Management Programming Guide for Cocoa*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

You must complete the object initialization (using an init method) before invoking release. For example, the following code shows an error:

```
id anObject = [MyObject alloc];
[anObject release];
```

You may call release from within an init method if initialization fails for some reason provided that you have at least called superclass's designated initializer.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `retainCount` (page 1312)

Declared In
`NSObject.h`

respondsToSelector:

Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message.

```
- (BOOL)respondsToSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that identifies a message.

Return Value
`YES` if the receiver implements or inherits a method that can respond to *aSelector*, otherwise `NO`.

Discussion
The application is responsible for determining whether a `NO` response should be considered an error.

You cannot test whether an object inherits a method from its superclass by sending `respondsToSelector:` to the object using the `super` keyword. This method will still be testing the object as a whole, not just the superclass's implementation. Therefore, sending `respondsToSelector:` to `super` is equivalent to sending it to `self`. Instead, you must invoke the `NSObject` class method `instancesRespondToSelector:` (page 791) directly on the object's superclass, as illustrated in the following code fragment.

```
if( [MySuperclass instancesRespondToSelector:@selector(aMethod)] ) {
    // invoke the inherited method
    [super aMethod];
}
```

You cannot simply use `[[self superclass] instancesRespondToSelector:@selector(aMethod)]` since this may cause the method to fail if it is invoked by a subclass.

Note that if the receiver is able to forward *aSelector* messages to another object, it will be able to respond to the message, albeit indirectly, even though this method returns `NO`.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `forwardInvocation:` (page 801) (`NSObject` class)
+ `instancesRespondToSelector:` (page 791) (`NSObject` class)

Declared In
`NSObject.h`

retain

Increments the receiver's reference count.

```
- (id)retain
```

Return Value
`self`.

Discussion
You send an object a `retain` message when you want to prevent it from being deallocated without your express permission.

An object is deallocated automatically when its reference count reaches 0. `retain` messages increment the reference count, and `release` (page 1310) messages decrement it. For more information on this mechanism, see *Memory Management Programming Guide for Cocoa*.

As a convenience, `retain` returns `self` because it is often used in nested expressions:

```
NSString *systemApps = [[NSString
    stringWithCString:"/Applications"] retain];
```

You would implement this method only if you were defining your own reference-counting scheme. Such implementations must return `self` and should not invoke the inherited method by sending a `retain` message to `super`.

Special Considerations
If garbage collection is enabled, this method is a no-op.

Availability
Available in iPhone OS 2.0 and later.

See Also
- `autorelease` (page 1303)
- `release` (page 1310)
- `retainCount` (page 1312)

Declared In
`NSObject.h`

retainCount

Returns the receiver's reference count.

```
- (NSUInteger)retainCount
```

Return Value
The receiver's reference count.

Discussion
You rarely send a `retainCount` message; however, you might implement this method in a class to implement your own reference-counting scheme. For objects that never get released (that is, their `release` (page 1310) method does nothing), this method should return `UINT_MAX`, as defined in `<limits.h>`.

The `retainCount` method does not account for any pending `autorelease` (page 1303) messages sent to the receiver.

This method is typically of limited value in debugging memory management issues.

Special Considerations

If garbage collection is enabled, the return value is undefined.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `autorelease` (page 1303)
- `retain` (page 1312)

Declared In

`NSObject.h`

self

Returns the receiver.

```
- (id)self
```

Return Value

The receiver.

Availability

Available in iPhone OS 2.0 and later.

See Also

- `class` (page 1304)

Declared In

`NSObject.h`

superclass

Returns the class object for the receiver's superclass.

```
- (Class)superclass
```

Return Value

The class object for the receiver's superclass.

Availability

Available in iPhone OS 2.0 and later.

See Also

+ `superclass` (page 796) (`NSObject` class)

Declared In

`NSObject.h`

zone

Returns a pointer to the zone from which the receiver was allocated.

```
- (NSZone *)zone
```

Return Value
A pointer to the zone from which the receiver was allocated.

Discussion
Objects created without specifying a zone are allocated from the default zone.

Availability
Available in iPhone OS 2.0 and later.

See Also
`+ allocWithZone:` (page 783) (`NSObject` class)

Declared In
`NSObject.h`

NSURLAuthenticationChallengeSender Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLAuthenticationChallenge.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSURLAuthenticationChallengeSender` protocol represents the interface that the sender of an authentication challenge must implement.

The methods in the protocol are generally sent by a delegate in response to receiving a `connection:didReceiveAuthenticationChallenge:` (page 1128) or `download:didReceiveAuthenticationChallenge:`. The different methods provide different ways of responding to authentication challenges.

Tasks

Protocol Methods

- `cancelAuthenticationChallenge:` (page 1316)
  Cancels a given authentication challenge.
- `continueWithoutCredentialForAuthenticationChallenge:` (page 1316)
  Attempt to continue downloading a request without providing a credential for a given challenge.
- `useCredential:forAuthenticationChallenge:` (page 1317)
  Attempt to use a given credential for a given authentication challenge.

Instance Methods

cancelAuthenticationChallenge:

Cancels a given authentication challenge.

```
- (void)cancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

challenge
  The authentication challenge to cancel.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLAuthenticationChallenge.h

continueWithoutCredentialForAuthenticationChallenge:

Attempt to continue downloading a request without providing a credential for a given challenge.

```
- (void)continueWithoutCredentialForAuthenticationChallenge:(NSURLAuthenticationChallenge
 *)challenge
```

Parameters

challenge
  A challenge without authentication credentials.

Discussion

This method has no effect if it is called with an authentication challenge that has already been handled.

Availability

Available in iPhone OS 2.0 and later.

Declared In
NSURLAuthenticationChallenge.h

useCredential:forAuthenticationChallenge:

Attempt to use a given credential for a given authentication challenge.

```
- (void)useCredential:(NSURLCredential *)credential
    forAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

credential
The credential to use for authentication.

challenge
The challenge for which to use *credential*.

Discussion
This method has no effect if it is called with an authentication challenge that has already been handled.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLAuthenticationChallenge.h

NSURLProtocolClient Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Availability:	Available in iPhone OS 2.0 and later.
Declared in:	Foundation/NSURLProtocol.h
Companion guide:	URL Loading System

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

The `NSURLProtocolClient` protocol provides the interface used by `NSURLProtocol` subclasses to communicate with the URL loading system. An application should never have the need to implement this protocol.

Tasks

Protocol Methods

`- URLProtocol:cachedResponseIsValid:` (page 1320)
Sent to indicate to the URL loading system that a cached response is valid.

- `URLProtocol:didCancelAuthenticationChallenge:` (page 1320)
  Sent to indicate to the URL loading system that an authentication challenge has been canceled.
- `URLProtocol:didFailWithError:` (page 1321)
  Sent when the load request fails due to an error.
- `URLProtocol:didLoadData:` (page 1321)
  An NSURLProtocol subclass instance, *protocol*, sends this message to `[protocol client]` as it loads *data*.
- `URLProtocol:didReceiveAuthenticationChallenge:` (page 1321)
  Sent to indicate to the URL loading system that an authentication challenge has been received.
- `URLProtocol:didReceiveResponse:cacheStoragePolicy:` (page 1322)
  Sent to indicate to the URL loading system that the protocol implementation has created a response object for the request.
- `URLProtocol:wasRedirectedToRequest:redirectResponse:` (page 1322)
  Sent to indicate to the URL loading system that the protocol implementation has been redirected.
- `URLProtocolDidFinishLoading:` (page 1323)
  Sent to indicate to the URL loading system that the protocol implementation has finished loading.

Instance Methods

URLProtocol:cachedResponseIsValid:

Sent to indicate to the URL loading system that a cached response is valid.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    cachedResponseIsValid:(NSCachedURLResponse *)cachedResponse
```

Parameters

protocol
The URL protocol object sending the message.

cachedResponse
The cached response whose validity is being communicated.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLProtocol.h

URLProtocol:didCancelAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been canceled.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

protocol
The URL protocol object sending the message.

challenge
The authentication challenge that was canceled.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

URLProtocol:didFailWithError:

Sent when the load request fails due to an error.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didFailWithError:(NSError *)error
```

Parameters

protocol
The URL protocol object sending the message.

error
The error that caused the failure of the load request.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

URLProtocol:didLoadData:

An NSURLProtocol subclass instance, *protocol*, sends this message to `[protocol client]` as it loads *data*.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didLoadData:(NSData *)data
```

Discussion
The data object must contain only new data loaded since the previous invocation of this method.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSURLProtocol.h

URLProtocol:didReceiveAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been received.

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

protocol
The URL protocol object sending the message.

challenge
The authentication challenge that has been received.

Discussion

The protocol client guarantees that it will answer the request on the same thread that called this method. The client may add a default credential to the challenge it issues to the connection delegate, if *protocol* did not provide one.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLProtocol.h

URLProtocol:didReceiveResponse:cacheStoragePolicy:

Sent to indicate to the URL loading system that the protocol implementation has created a response object for the request.

```
- (void)URLProtocol:(NSURLProtocol *)protocol didReceiveResponse:(NSURLResponse
    *)response cacheStoragePolicy:(NSURLCacheStoragePolicy)policy
```

Parameters

protocol
The URL protocol object sending the message.

response
The newly available response object.

policy
The cache storage policy for the response.

Discussion

The implementation should provide the NSURLCacheStoragePolicy that should be used if the response is to be stored in a cache as the *policy* value.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLProtocol.h

URLProtocol:wasRedirectedToRequest:redirectResponse:

Sent to indicate to the URL loading system that the protocol implementation has been redirected.

```
- (void)URLProtocol:(NSURLProtocol *)protocol wasRedirectedToRequest:(NSURLRequest
    *)request redirectResponse:(NSURLResponse *)redirectResponse
```

Parameters

protocol

The URL protocol object sending the message.

request

The new request that the original request was redirected to.

redirectResponse

The response from the original request that caused the redirect.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLProtocol.h

URLProtocolDidFinishLoading:

Sent to indicate to the URL loading system that the protocol implementation has finished loading.

```
- (void)URLProtocolDidFinishLoading:(NSURLProtocol *)protocol
```

Parameters

protocol

The URL protocol object sending the message.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSURLProtocol.h

Functions

Foundation Functions Reference

Framework: Foundation/Foundation.h

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

This chapter describes the functions and function-like macros defined in the Foundation Framework.

Functions by Task

Assertions

For additional information about Assertions, see *Assertions and Logging*.

`NSAssert` (page 1336)
Generates an assertion if a given condition is false.

`NSAssert1` (page 1336)
Generates an assertion if a given condition is false.

`NSAssert2` (page 1337)
Generates an assertion if a given condition is false.

`NSAssert3` (page 1338)
Generates an assertion if a given condition is false.

NSAssert4 (page 1339)
Generates an assertion if a given condition is false.

NSAssert5 (page 1340)
Generates an assertion if a given condition is false.

NSCAssert (page 1341)
Generates an assertion if the given condition is false.

NSCAssert1 (page 1342)
NSCAssert1 is one of a series of macros that generate assertions if the given condition is false.

NSCAssert2 (page 1342)
NSCAssert2 is one of a series of macros that generate assertions if the given condition is false.

NSCAssert3 (page 1343)
NSCAssert3 is one of a series of macros that generate assertions if the given condition is false.

NSCAssert4 (page 1344)
NSCAssert4 is one of a series of macros that generate assertions if the given condition is false.

NSCAssert5 (page 1344)
NSCAssert5 is one of a series of macros that generate assertions if the given condition is false.

NSCParameterAssert (page 1348)
Evaluates the specified parameter.

NSParameterAssert (page 1368)
Validates the specified parameter.

Bundles

For additional information on generating strings files see “Generating Strings Files”.

NSLocalizedString (page 1362)
Returns a localized version of a string.

NSLocalizedStringFromTable (page 1362)
Returns a localized version of a string.

NSLocalizedStringFromTableInBundle (page 1363)
Returns a localized version of a string.

NSLocalizedStringWithDefaultValue (page 1363)
Returns a localized version of a string.

Byte Ordering

NSConvertHostDoubleToSwapped (page 1345)
Performs a type conversion.

NSConvertHostFloatToSwapped (page 1346)
Performs a type conversion.

NSConvertSwappedDoubleToHost (page 1346)
Performs a type conversion.

NSConvertSwappedFloatToHost (page 1347)
Performs a type conversion.

NSHostByteOrder (page 1361)
Returns the endian format.

NSSwapBigDoubleToHost (page 1377)
A utility for swapping the bytes of a number.

NSSwapBigFloatToHost (page 1378)
A utility for swapping the bytes of a number.

NSSwapBigIntToHost (page 1378)
A utility for swapping the bytes of a number.

NSSwapBigLongLongToHost (page 1379)
A utility for swapping the bytes of a number.

NSSwapBigLongToHost (page 1379)
A utility for swapping the bytes of a number.

NSSwapBigShortToHost (page 1380)
A utility for swapping the bytes of a number.

NSSwapDouble (page 1380)
A utility for swapping the bytes of a number.

NSSwapFloat (page 1380)
A utility for swapping the bytes of a number.

NSSwapHostDoubleToBig (page 1381)
A utility for swapping the bytes of a number.

NSSwapHostDoubleToLittle (page 1381)
A utility for swapping the bytes of a number.

NSSwapHostFloatToBig (page 1382)
A utility for swapping the bytes of a number.

NSSwapHostFloatToLittle (page 1382)
A utility for swapping the bytes of a number.

NSSwapHostIntToBig (page 1383)
A utility for swapping the bytes of a number.

NSSwapHostIntToLittle (page 1383)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToBig (page 1384)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToLittle (page 1384)
A utility for swapping the bytes of a number.

NSSwapHostLongToBig (page 1385)
A utility for swapping the bytes of a number.

NSSwapHostLongToLittle (page 1385)
A utility for swapping the bytes of a number.

NSSwapHostShortToBig (page 1386)
A utility for swapping the bytes of a number.

NSSwapHostShortToLittle (page 1386)
A utility for swapping the bytes of a number.

NSSwapInt (page 1387)
A utility for swapping the bytes of a number.

NSSwapLittleDoubleToHost (page 1387)
A utility for swapping the bytes of a number.

NSSwapLittleFloatToHost (page 1388)
A utility for swapping the bytes of a number.

NSSwapLittleIntToHost (page 1388)
A utility for swapping the bytes of a number.

NSSwapLittleLongLongToHost (page 1389)
A utility for swapping the bytes of a number.

NSSwapLittleLongToHost (page 1389)
A utility for swapping the bytes of a number.

NSSwapLittleShortToHost (page 1390)
A utility for swapping the bytes of a number.

NSSwapLong (page 1390)
A utility for swapping the bytes of a number.

NSSwapLongLong (page 1391)
A utility for swapping the bytes of a number.

NSSwapShort (page 1391)
A utility for swapping the bytes of a number.

Decimals

The class "NSDecimalNumber" (page 275) may also be used for decimal arithmetic.

NSDecimalAdd (page 1350)
Adds two decimal values.

NSDecimalCompact (page 1351)
Compacts the decimal structure for efficiency.

NSDecimalCompare (page 1351)
Compares two decimal values.

NSDecimalCopy (page 1352)
Copies the value of a decimal number.

NSDecimalDivide (page 1352)
Divides one decimal value by another.

NSDecimalIsNotANumber (page 1353)
Returns a Boolean that indicates whether a given decimal contains a valid number.

NSDecimalMultiply (page 1353)
Multiplies two decimal numbers together.

NSDecimalMultiplyByPowerOf10 (page 1353)
Multiplies a decimal by the specified power of 10.

NSDecimalNormalize (page 1354)
Normalizes the internal format of two decimal numbers to simplify later operations.

NSDecimalPower (page 1355)
Raises the decimal value to the specified power.

NSDecimalRound (page 1355)
Rounds off the decimal value.

NSDecimalString (page 1356)
Returns a string representation of the decimal value.

NSDecimalSubtract (page 1356)
Subtracts one decimal value from another.

Exception Handling

You can find the following macros implemented in NSException.h. *Exception Programming Topics for Cocoa* discusses these macros and gives examples of their usage. These macros are useful for code that needs to run on versions of the system prior to Mac OS X v10.3 For later versions of the operating system, you should use the Objective-C compiler directives @try, @catch, @throw, and @finally; for information about these directives, see "Exception Handling and Thread Synchronization" in *The Objective-C 2.0 Programming Language*.

NS_DURING (page 1396)
Marks the start of the exception-handling domain.

NS_ENDHANDLER (page 1396)
Marks the end of the local event handler.

NS_HANDLER (page 1397)
Marks the end of the exception-handling domain and the start of the local exception handler.

NS_VALUERETURN (page 1397)
Permits program control to exit from an exception-handling domain with a value of a specified type.

NS_VOIDRETURN (page 1397)
Permits program control to exit from an exception-handling domain.

Managing Object Allocation and Deallocation

NSAllocateObject (page 1335)
Creates and returns a new instance of a given class.

NSCopyObject (page 1347)
Creates an exact copy of an object.

NSDeallocateObject (page 1350)
Destroys an existing object.

NSDecrementExtraRefCountWasZero (page 1356)
Decrements the specified object's reference count.

NSExtraRefCount (page 1358)
Returns the specified object's reference count.

NSIncrementExtraRefCount (page 1361)
Increments the specified object's reference count.

NSShouldRetainWithZone (page 1374)
Indicates whether an object should be retained.

Interacting with the Objective-C Runtime

`NSGetSizeAndAlignment` (page 1359)
Obtains the actual size and the aligned size of an encoded type.

`NSClassFromString` (page 1345)
Obtains a class by name.

`NSStringFromClass` (page 1376)
Returns the name of a class as a string.

`NSSelectorFromString` (page 1373)
Returns the selector with a given name.

`NSStringFromSelector` (page 1377)
Returns a string representation of a given selector.

`NSStringFromProtocol` (page 1376)
Returns the name of a protocol as a string.

`NSProtocolFromString` (page 1369)
Returns a the protocol with a given name.

Logging Output

`NSLog` (page 1364)
Logs error an message to `stderr`.

`NSLogv` (page 1365)
Logs an error message to `stderr`.

Managing File Paths

`NSFullUserName` (page 1358)
Returns a string containing the full name of the current user.

`NSHomeDirectory` (page 1360)
Returns the path to the current user's home directory.

`NSHomeDirectoryForUser` (page 1360)
Returns the path to a given user's home directory.

`NSOpenStepRootDirectory` (page 1367)
Returns the root directory of the user's system.

`NSSearchPathForDirectoriesInDomains` (page 1372)
Creates a list of directory search paths.

`NSTemporaryDirectory` (page 1392)
Returns the path of the temporary directory for the current user.

`NSUserName` (page 1393)
Returns the logon name of the current user.

Managing Points

NSPointFromCGPoint (page 1368)
Returns an NSPoint typecast from a CGPoint.

NSPointToCGPoint (page 1369)
Returns a CGPoint typecast from an NSPoint.

Manipulating Ranges

NSEqualRanges (page 1358)
Returns a Boolean value that indicates whether two given ranges are equal.

NSIntersectionRange (page 1361)
Returns the intersection of the specified ranges.

NSLocationInRange (page 1364)
Returns a Boolean value that indicates whether a specified position is in a given range.

NSMakeRange (page 1366)
Creates a new NSRange from the specified values.

NSMaxRange (page 1367)
Returns the number 1 greater than the maximum value within the range.

NSRangeFromString (page 1370)
Returns a range from a text-based representation.

NSStringFromRange (page 1377)
Returns a string representation of a range.

NSUnionRange (page 1392)
Returns the intersection of the specified ranges.

Manipulating Rectangles

NSRectFromCGRect (page 1370)
Returns an NSRect typecast from a CGRect.

NSRectToCGRect (page 1371)
Returns a CGRect typecast from an NSRect.

Sizes

NSSizeFromCGSize (page 1375)
Returns an NSSize typecast from a CGSize.

NSSizeToCGSize (page 1375)
Returns a CGSize typecast from an NSSize.

Uncaught Exception Handlers

Whether there's an uncaught exception handler function, any uncaught exceptions cause the program to terminate, unless the exception is raised during the posting of a notification.

`NSGetUncaughtExceptionHandler` (page 1359)
Returns the top-level error handler.

`NSSetUncaughtExceptionHandler` (page 1374)
Changes the top-level error handler.

Managing Memory

`NSDefaultMallocZone` (page 1357)
Returns the default zone.

`NSMakeCollectable` (page 1366)
Makes a newly allocated Core Foundation object eligible for collection.

`NSAllocateMemoryPages` (page 1335)
Allocates a new block of memory.

`NSCopyMemoryPages` (page 1347)
Copies a block of memory.

`NSDeallocateMemoryPages` (page 1349)
Deallocates the specified block of memory.

`NSLogPageSize` (page 1365)
Returns the binary log of the page size.

`NSPageSize` (page 1367)
Returns the number of bytes in a page.

`NSRealMemoryAvailable` (page 1370)
Returns information about the user's system.

`NSRoundDownToMultipleOfPageSize` (page 1372)
Returns the specified number of bytes rounded down to a multiple of the page size.

`NSRoundUpToMultipleOfPageSize` (page 1372)
Returns the specified number of bytes rounded up to a multiple of the page size.

Managing Zones

`NSCreateZone` (page 1349)
Creates a new zone.

`NSRecycleZone` (page 1371)
Frees memory in a zone.

`NSSetZoneName` (page 1374)
Sets the name of the specified zone.

`NSZoneCalloc` (page 1393)
Allocates memory in a zone.

`NSZoneFree` (page 1394)
Deallocates a block of memory in the specified zone.

`NSZoneFromPointer` (page 1394)
Gets the zone for a given block of memory.

`NSZoneMalloc` (page 1395)
Allocates memory in a zone.

`NSZoneName` (page 1395)
Returns the name of the specified zone.

`NSZoneRealloc` (page 1396)
Allocates memory in a zone.

Functions

NSAllocateMemoryPages

Allocates a new block of memory.

```
void * NSAllocateMemoryPages (
   NSUInteger bytes
);
```

Discussion

Allocates the integral number of pages whose total size is closest to, but not less than, *byteCount*. The allocated pages are guaranteed to be filled with zeros. If the allocation fails, raises `NSInvalidArgumentException`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSCopyMemoryPages` (page 1347)

`NSDeallocateMemoryPages` (page 1349)

Declared In

`NSZone.h`

NSAllocateObject

Creates and returns a new instance of a given class.

```
id NSAllocateObject (
   Class aClass,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters

aClass
The class of which to create an instance.

extraBytes
The number of extra bytes required for indexed instance variables (this value is typically `0`).

zone
The zone in which to create the new instance (pass `NULL` to specify the default zone).

Return Value

A new instance of *aClass*) or `nil` if an instance could not be created.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSCopyObject (page 1347)
NSDeallocateObject (page 1350)

Declared In
NSObject.h

NSAssert

Generates an assertion if a given condition is false.

```
#define NSAssert(condition, desc)
```

Parameters

condition
An expression that evaluates to YES or NO.

desc
An NSString object that contains an error message describing the failure condition.

Discussion
The NSAssert macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an NSInternalInconsistencyException exception. If *condition* evaluates to NO, the macro invokes handleFailureInMethod:object:file:lineNumber:description: (page 71) on the assertion handler for the current thread, passing *desc* as the description string.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined.

See Also
NSLog (page 1364)
NSLogv (page 1365)
NSAssert1 (page 1336)
NSCAssert (page 1341)
NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSAssert1

Generates an assertion if a given condition is false.

```
#define NSAssert1(condition, desc, arg1)
```

Parameters

condition

An expression that evaluates to YES or NO.

desc

An NSString object that contains a printf-style string containing an error message describing the failure condition and a placeholder for a single argument.

arg1

An argument to be inserted, in place, into *desc*.

Discussion

The NSAssert1 macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an NSInternalInconsistencyException exception. If *condition* evaluates to NO, the macro invokes handleFailureInMethod:object:file:lineNumber:description: (page 71) on the assertion handler for the current thread, passing *desc* as the description string and *arg1* as a substitution variable.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined.

See Also

NSLog (page 1364)
NSLogv (page 1365)
NSAssert (page 1336)
NSAssert2 (page 1337)
NSAssert3 (page 1338)
NSAssert4 (page 1339)
NSAssert5 (page 1340)
NSCAssert (page 1341)
NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSAssert2

Generates an assertion if a given condition is false.

```
#define NSAssert2(condition, desc, arg1, arg2)
```

Parameters

condition

An expression that evaluates to YES or NO.

desc

An NSString object that contains a printf-style string containing an error message describing the failure condition and placeholders for two arguments.

arg1
An argument to be inserted, in place, into *desc*.

arg2
An argument to be inserted, in place, into *desc*.

Discussion

The `NSAssert2` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *condition* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 71) on the assertion handler for the current thread, passing *desc* as the description string and *arg1* and *arg2* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

See Also

`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSAssert` (page 1336)
`NSAssert1` (page 1336)
`NSAssert3` (page 1338)
`NSAssert4` (page 1339)
`NSAssert5` (page 1340)
`NSCAssert` (page 1341)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSAssert3

Generates an assertion if a given condition is false.

```
#define NSAssert3(condition, desc, arg1, arg2, arg3)
```

Parameters

condition
An expression that evaluates to `YES` or `NO`.

desc
An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for three arguments.

arg1
An argument to be inserted, in place, into *desc*.

arg2
An argument to be inserted, in place, into *desc*.

arg3
An argument to be inserted, in place, into *desc*.

Discussion

The `NSAssert3` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *`condition`* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 71) on the assertion handler for the current thread, passing *`desc`* as the description string and *`arg1`*, *`arg2`*, and *`arg3`* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

See Also

`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSAssert` (page 1336)
`NSAssert1` (page 1336)
`NSAssert2` (page 1337)
`NSAssert4` (page 1339)
`NSAssert5` (page 1340)
`NSCAssert` (page 1341)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSAssert4

Generates an assertion if a given condition is false.

```
#define NSAssert4(condition, desc, arg1, arg2, arg3, arg4)
```

Parameters

`condition`

An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for four arguments.

`arg1`

An argument to be inserted, in place, into *`desc`*.

`arg2`

An argument to be inserted, in place, into *`desc`*.

`arg3`

An argument to be inserted, in place, into *`desc`*.

`arg4`

An argument to be inserted, in place, into *`desc`*.

Discussion

The `NSAssert4` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *`condition`* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 71) on the assertion handler for the current thread, passing *`desc`* as the description string and *`arg1`*, *`arg2`*, *`arg3`*, and *`arg4`* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

See Also

`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSAssert` (page 1336)
`NSAssert1` (page 1336)
`NSAssert2` (page 1337)
`NSAssert3` (page 1338)
`NSAssert5` (page 1340)
`NSCAssert` (page 1341)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSAssert5

Generates an assertion if a given condition is false.

```
#define NSAssert5(condition, desc, arg1, arg2, arg3, arg4, arg5)
```

Parameters

`condition`
An expression that evaluates to `YES` or `NO`.

`desc`
An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for five arguments.

`arg1`
An argument to be inserted, in place, into *`desc`*.

`arg2`
An argument to be inserted, in place, into *`desc`*.

`arg3`
An argument to be inserted, in place, into *`desc`*.

`arg4`
An argument to be inserted, in place, into *`desc`*.

`arg5`
An argument to be inserted, in place, into *`desc`*.

Discussion

The `NSAssert5` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If *`condition`* evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` (page 71) on the assertion handler for the current thread, passing *`desc`* as the description string and *`arg1`*, *`arg2`*, *`arg3`*, *`arg4`*, and *`arg5`* as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

See Also
`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSAssert` (page 1336)
`NSAssert1` (page 1336)
`NSAssert2` (page 1337)
`NSAssert3` (page 1338)
`NSAssert4` (page 1339)
`NSCAssert` (page 1341)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSCAssert

Generates an assertion if the given condition is false.

```
NSCAssert(condition, NSString *description)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions. `NSCAssert` takes no arguments other than the condition and format string.

The *`condition`* must be an expression that evaluates to true or false. *`description`* is a printf-style format string that describes the failure condition.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

See Also
`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSAssert` (page 1336)
`NSCAssert1` (page 1342)

NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSCAssert1

NSCAssert1 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert1(condition, NSString *description, arg1)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert1 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *condition* expression must evaluate to true or false. *description* is a printf-style format string that describes the failure condition. *arg1* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

See Also

NSLog (page 1364)
NSLogv (page 1365)
NSCAssert (page 1341)
NSCAssert2 (page 1342)
NSCAssert3 (page 1343)
NSCAssert4 (page 1344)
NSCAssert5 (page 1344)
NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSCAssert2

NSCAssert2 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert2(condition, NSString *description, arg1, arg2)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert2 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *condition* expression must evaluate to true or false. *description* is a printf-style format string that describes the failure condition. Each *argn* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

See Also
NSLog (page 1364)
NSLogv (page 1365)
NSCAssert (page 1341)
NSCAssert1 (page 1342)
NSCAssert3 (page 1343)
NSCAssert4 (page 1344)
NSCAssert5 (page 1344)
NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSCAssert3

NSCAssert3 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert3(condition, NSString *description, arg1, arg2, arg3)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert3 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *condition* expression must evaluate to true or false. *description* is a printf-style format string that describes the failure condition. Each *argn* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

See Also
NSLog (page 1364)
NSLogv (page 1365)
NSCAssert (page 1341)
NSCAssert1 (page 1342)
NSCAssert2 (page 1342)
NSCAssert4 (page 1344)
NSCAssert5 (page 1344)
NSCParameterAssert (page 1348)
NSParameterAssert (page 1368)

NSCAssert4

`NSCAssert4` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert4(condition, NSString *description, arg1, arg2, arg3, arg4)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert4` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. Each *`argn`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

See Also

`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSCAssert` (page 1341)
`NSCAssert1` (page 1342)
`NSCAssert2` (page 1342)
`NSCAssert3` (page 1343)
`NSCAssert5` (page 1344)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSCAssert5

`NSCAssert5` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert5(condition, NSString *description, arg1, arg2, arg3, arg4, arg5)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert5` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The *`condition`* expression must evaluate to true or false. *`description`* is a printf-style format string that describes the failure condition. Each *`argn`* is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

See Also

`NSLog` (page 1364)
`NSLogv` (page 1365)
`NSCAssert` (page 1341)
`NSCAssert1` (page 1342)
`NSCAssert2` (page 1342)
`NSCAssert3` (page 1343)
`NSCAssert4` (page 1344)
`NSCParameterAssert` (page 1348)
`NSParameterAssert` (page 1368)

NSClassFromString

Obtains a class by name.

```
Class NSClassFromString (
   NSString *aClassName
);
```

Parameters

aClassName
The name of a class.

Return Value

The class object named by *aClassName*, or `nil` if no class by that name is currently loaded. If *aClassName* is `nil`, returns `nil`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSStringFromClass` (page 1376)
`NSProtocolFromString` (page 1369)
`NSSelectorFromString` (page 1373)

Declared In

`NSObjCRuntime.h`

NSConvertHostDoubleToSwapped

Performs a type conversion.

```
NSSwappedDouble NSConvertHostDoubleToSwapped (
   double x
);
```

Discussion

Converts the double value in *x* to a value whose bytes can be swapped. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapHostDoubleToBig (page 1381)
NSSwapHostDoubleToLittle (page 1381)

Declared In
NSByteOrder.h

NSConvertHostFloatToSwapped

Performs a type conversion.

```
NSSwappedFloat NSConvertHostFloatToSwapped (
   float x
);
```

Discussion
Converts the float value in *x* to a value whose bytes can be swapped. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapHostFloatToBig (page 1382)
NSSwapHostFloatToLittle (page 1382)

Declared In
NSByteOrder.h

NSConvertSwappedDoubleToHost

Performs a type conversion.

```
double NSConvertSwappedDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the value in *x* to a double value. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapBigDoubleToHost (page 1377)
NSSwapLittleDoubleToHost (page 1387)

Declared In
NSByteOrder.h

NSConvertSwappedFloatToHost

Performs a type conversion.

```
float NSConvertSwappedFloatToHost (
   NSSwappedFloat x
);
```

Discussion

Converts the value in *x* to a float value. This function does not actually swap the bytes of *x*. You should not need to call this function directly.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSSwapBigFloatToHost (page 1378)

NSSwapLittleFloatToHost (page 1388)

Declared In

NSByteOrder.h

NSCopyMemoryPages

Copies a block of memory.

```
void NSCopyMemoryPages (
   const void *source,
   void *dest,
   NSUInteger bytes
);
```

Discussion

Copies (or copies on write) *byteCount* bytes from *source* to *destination*.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSAllocateMemoryPages (page 1335)

NSDeallocateMemoryPages (page 1349)

Declared In

NSZone.h

NSCopyObject

Creates an exact copy of an object.

```
id NSCopyObject (
   id object,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters

object

The object to copy.

extraBytes

The number of extra bytes required for indexed instance variables (this value is typically 0).

zone

The zone in which to create the new instance (pass NULL to specify the default zone).

Return Value

A new object that's an exact copy of *anObject*, or nil if *object* is nil or if *object* could not be copied.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSAllocateObject (page 1335)
NSDeallocateObject (page 1350)

Declared In

NSObject.h

NSCParameterAssert

Evaluates the specified parameter.

```
NSCParameterAssert(condition)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

This macro validates a parameter for a C function. Simply provide the parameter as the condition argument. The macro evaluates the parameter and, if the parameter evaluates to false, logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

See Also

NSLog (page 1364)
NSLogv (page 1365)
NSAssert (page 1336)
NSCAssert (page 1341)

NSParameterAssert (page 1368)

NSCreateZone

Creates a new zone.

```
NSZone * NSCreateZone (
   NSUInteger startSize,
   NSUInteger granularity,
   BOOL canFree
);
```

Return Value

A pointer to a new zone of *startSize* bytes, which will grow and shrink by *granularity* bytes. If *canFree* is 0, the allocator will never free memory, and `malloc` will be fast. Returns `NULL` if a new zone could not be created.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSDefaultMallocZone (page 1357)
NSRecycleZone (page 1371)
NSSetZoneName (page 1374)

Declared In

NSZone.h

NSDeallocateMemoryPages

Deallocates the specified block of memory.

```
void NSDeallocateMemoryPages (
   void *ptr,
   NSUInteger bytes
);
```

Discussion

This function deallocates memory that was allocated with `NSAllocateMemoryPages`.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSCopyMemoryPages (page 1347)
NSAllocateMemoryPages (page 1335)

Declared In

NSZone.h

NSDeallocateObject

Destroys an existing object.

```
void NSDeallocateObject (
   id object
);
```

Parameters

object

An object.

Discussion

This function deallocates *object*, which must have been allocated using `NSAllocateObject`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSCopyObject` (page 1347)

`NSAllocateObject` (page 1335)

Declared In

`NSObject.h`

NSDecimalAdd

Adds two decimal values.

```
NSCalculationError NSDecimalAdd (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion

Adds *leftOperand* to *rightOperand* and stores the sum in *result*.

An NSDecimal can represent a number with up to 38 significant digits. If a number is more precise than that, it must be rounded off. *roundingMode* determines how to round it off. There are four possible rounding modes:

`NSRoundDown`	Round return values down.
`NSRoundUp`	Round return values up.
`NSRoundPlain`	Round to the closest possible return value; when caught halfway between two positive numbers, round up; when caught between two negative numbers, round down.
`NSRoundBankers`	Round to the closest possible return value; when halfway between two possibilities, return the possibility whose last digit is even.

The return value indicates whether any machine limitations were encountered in the addition. If none were encountered, the function returns `NSCalculationNoError`. Otherwise it may return one of the following values: `NSCalculationLossOfPrecision`, `NSCalculationOverflow` or `NSCalculationUnderflow`. For descriptions of all these error conditions, see `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1252) in NSDecimalNumberBehaviors.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimal.h`

NSDecimalCompact

Compacts the decimal structure for efficiency.

```
void NSDecimalCompact (
   NSDecimal *number
);
```

Discussion
Formats number so that calculations using it will take up as little memory as possible. All the `NSDecimal...` arithmetic functions expect compact NSDecimal arguments.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimal.h`

NSDecimalCompare

Compares two decimal values.

```
NSComparisonResult NSDecimalCompare (
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand
);
```

Return Value
`NSOrderedDescending` if *`leftOperand`* is bigger than *`rightOperand`*; `NSOrderedAscending` if *`rightOperand`* is bigger than *`leftOperand`*; or `NSOrderedSame` if the two operands are equal.

Discussion
For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalCopy

Copies the value of a decimal number.

```
void NSDecimalCopy (
   NSDecimal *destination,
   const NSDecimal *source
);
```

Discussion
Copies the value in *source* to *destination*.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalDivide

Divides one decimal value by another.

```
NSCalculationError NSDecimalDivide (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Divides *leftOperand* by *rightOperand* and stores the quotient, possibly rounded off according to *roundingMode*, in *result*. If *rightOperand* is 0, returns NSDivideByZero.

For explanations of the possible return values and rounding modes, see NSDecimalAdd (page 1350).

Note that repeating decimals or numbers with a mantissa larger than 38 digits cannot be represented precisely.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalIsNotANumber

Returns a Boolean that indicates whether a given decimal contains a valid number.

```
BOOL NSDecimalIsNotANumber (
   const NSDecimal *dcm
);
```

Return Value

`YES` if the value in *`decimal`* represents a valid number, otherwise `NO`.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimal.h`

NSDecimalMultiply

Multiplies two decimal numbers together.

```
NSCalculationError NSDecimalMultiply (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion

Multiplies *`rightOperand`* by *`leftOperand`* and stores the product, possibly rounded off according to *`roundingMode`*, in *`result`*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 1350).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSDecimal.h`

NSDecimalMultiplyByPowerOf10

Multiplies a decimal by the specified power of 10.

```
NSCalculationError NSDecimalMultiplyByPowerOf10 (
   NSDecimal *result,
   const NSDecimal *number,
   short power,
   NSRoundingMode roundingMode
);
```

Discussion
Multiplies *number* by *power* of 10 and stores the product, possibly rounded off according to *roundingMode*, in *result*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 1350).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimal.h`

NSDecimalNormalize

Normalizes the internal format of two decimal numbers to simplify later operations.

```
NSCalculationError NSDecimalNormalize (
   NSDecimal *number1,
   NSDecimal *number2,
   NSRoundingMode roundingMode
);
```

Discussion
An NSDecimal is represented in memory as a mantissa and an exponent, expressing the value mantissa x 10exponent. A number can have many NSDecimal representations; for example, the following table lists several valid NSDecimal representations for the number 100:

Mantissa	Exponent
100	0
10	1
1	2

Format *number1* and *number2* so that they have equal exponents. This format makes addition and subtraction very convenient. Both `NSDecimalAdd` (page 1350) and `NSDecimalSubtract` (page 1356) call `NSDecimalNormalize`. You may want to use it if you write more complicated addition or subtraction routines.

For explanations of the possible return values, see `NSDecimalAdd` (page 1350).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalPower

Raises the decimal value to the specified power.

```
NSCalculationError NSDecimalPower (
   NSDecimal *result,
   const NSDecimal *number,
   NSUInteger power,
   NSRoundingMode roundingMode
);
```

Discussion
Raises *number* to *power*, possibly rounding off according to *roundingMode*, and stores the resulting value in *result*.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 1350).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalRound

Rounds off the decimal value.

```
void NSDecimalRound (
   NSDecimal *result,
   const NSDecimal *number,
   NSInteger scale,
   NSRoundingMode roundingMode
);
```

Discussion
Rounds *number* off according to the parameters *scale* and *roundingMode* and stores the result in *result*.

The *scale* value specifies the number of digits *result* can have after its decimal point. *roundingMode* specifies the way that number is rounded off. There are four possible values for *roundingMode*: `NSRoundDown`, `NSRoundUp`, `NSRoundPlain`, and `NSRoundBankers`. For thorough discussions of *scale* and *roundingMode*, see NSDecimalNumberBehaviors.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalString

Returns a string representation of the decimal value.

```
NSString * NSDecimalString (
   const NSDecimal *dcm,
   id locale
);
```

Discussion
Returns a string representation of *decimal*. *locale* determines the format of the decimal separator.

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecimalSubtract

Subtracts one decimal value from another.

```
NSCalculationError NSDecimalSubtract (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Subtracts *rightOperand* from *leftOperand* and stores the difference, possibly rounded off according to *roundingMode*, in *result*.

For explanations of the possible return values and rounding modes, see NSDecimalAdd (page 1350).

For more information, see *Number and Value Programming Topics for Cocoa*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSDecimal.h

NSDecrementExtraRefCountWasZero

Decrements the specified object's reference count.

```
BOOL NSDecrementExtraRefCountWasZero (
   id object
);
```

Parameters

object

An object.

Return Value

`NO` if *anObject* had an extra reference count, or `YES` if *anObject* didn't have an extra reference count—indicating that the object should be deallocated (with `dealloc`).

Discussion

Decrements the "extra reference" count of *anObject*. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the `retain` (page 1312) or `release` (page 1310) methods.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSExtraRefCount` (page 1358)

`NSIncrementExtraRefCount` (page 1361)

Declared In

`NSObject.h`

NSDefaultMallocZone

Returns the default zone.

```
NSZone * NSDefaultMallocZone (
   void
);
```

Return Value

The default zone, which is created automatically at startup.

Discussion

This zone is used by the standard C function `malloc`.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSCreateZone` (page 1349)

Declared In

`NSZone.h`

NSEqualRanges

Returns a Boolean value that indicates whether two given ranges are equal.

```
BOOL NSEqualRanges (
   NSRange range1,
   NSRange range2
);
```

Return Value

`YES` if *`range1`* and *`range2`* have the same locations and lengths.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRange.h`

NSExtraRefCount

Returns the specified object's reference count.

```
NSUInteger NSExtraRefCount (
   id object
);
```

Parameters

`object`

An object.

Return Value

The current reference count of *`object`*.

Discussion

This function is used in conjunction with `NSIncrementExtraRefCount` (page 1361) and `NSDecrementExtraRefCountWasZero` (page 1356) in situations where you need to override an object's `retain` (page 1312) and `release` (page 1310) methods.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSObject.h`

NSFullUserName

Returns a string containing the full name of the current user.

```
NSString * NSFullUserName (
   void
);
```

Return Value

A string containing the full name of the current user.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSUserName (page 1393)

Declared In
NSPathUtilities.h

NSGetSizeAndAlignment

Obtains the actual size and the aligned size of an encoded type.

```
const char * NSGetSizeAndAlignment (
   const char *typePtr,
   NSUInteger *sizep,
   NSUInteger *alignp
);
```

Discussion
Obtains the actual size and the aligned size of the first data type represented by *typePtr* and returns a pointer to the position of the next data type in *typePtr*. You can specify NULL for either *sizep* or *alignp* to ignore the corresponding information.

The value returned in *alignp* is the aligned size of the data type; for example, on some platforms, the aligned size of a char might be 2 bytes while the actual physical size is 1 byte.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSObjCRuntime.h

NSGetUncaughtExceptionHandler

Returns the top-level error handler.

```
NSUncaughtExceptionHandler * NSGetUncaughtExceptionHandler (
   void
);
```

Return Value
A pointer to the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSetUncaughtExceptionHandler (page 1374)

Declared In
NSException.h

NSHomeDirectory

Returns the path to the current user's home directory.

```
NSString * NSHomeDirectory (
   void
);
```

Return Value

The path to the current user's home directory.

Discussion

For more information on file-system utilities, see *Low-Level File Management Programming Topics*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSFullUserName` (page 1358)
`NSUserName` (page 1393)
`NSHomeDirectoryForUser` (page 1360)

Declared In

`NSPathUtilities.h`

NSHomeDirectoryForUser

Returns the path to a given user's home directory.

```
NSString * NSHomeDirectoryForUser (
   NSString *userName
);
```

Parameters

`userName`

The name of a user.

Return Value

The path to the home directory for the user specified by *`userName`*.

Discussion

For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSFullUserName` (page 1358)
`NSUserName` (page 1393)
`NSHomeDirectory` (page 1360)

Declared In

`NSPathUtilities.h`

NSHostByteOrder

Returns the endian format.

```
long NSHostByteOrder (
   void
);
```

Return Value

The endian format, either `NS_LittleEndian` or `NS_BigEndian`.

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSByteOrder.h`

NSIncrementExtraRefCount

Increments the specified object's reference count.

```
void NSIncrementExtraRefCount (
   id object
);
```

Parameters

object

An object.

Discussion

This function increments the "extra reference" count of *object*. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the retain or release methods.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSExtraRefCount` (page 1358)
`NSDecrementExtraRefCountWasZero` (page 1356)

Declared In

`NSObject.h`

NSIntersectionRange

Returns the intersection of the specified ranges.

```
NSRange NSIntersectionRange (
   NSRange range1,
   NSRange range2
);
```

Return Value
A range describing the intersection of *range1* and *range2*—that is, a range containing the indices that exist in both ranges.

Discussion
If the returned range's length field is 0, then the two ranges don't intersect, and the value of the location field is undefined.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSUnionRange (page 1392)

Declared In
NSRange.h

NSLocalizedString

Returns a localized version of a string.

```
NSString *NSLocalizedString(NSString *key, NSString *comment)
```

Return Value
The result of invoking localizedStringForKey:value:table: (page 95) on the main bundle and a nil table.

Discussion
In order to be parsed correctly by genstrings, *key* should not contain any high-ASCII characters. You can specify Unicode characters in *key* using \\Uxxxx—see the -u option for for the genstrings utility.

For more information, see NSBundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

NSLocalizedStringFromTable

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTable(NSString *key, NSString *tableName, NSString
 *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 95) on the main bundle, passing it the specified *key* and *tableName*.

Discussion
In order to be parsed correctly by `genstrings`, *key* must not contain any high-ASCII characters. You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see NSBundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

NSLocalizedStringFromTableInBundle

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTableInBundle(NSString *key, NSString *tableName,
NSBundle *bundle, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 95) on *bundle*, passing it the specified *key* and *tableName*.

Discussion
In order to be parsed correctly by `genstrings`, *key* must not contain any high-ASCII characters. You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see NSBundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSBundle.h`

NSLocalizedStringWithDefaultValue

Returns a localized version of a string.

```
NSString NSLocalizedStringWithDefaultValue(NSString *key, NSString *tableName,
NSBundle *bundle, NSString *value, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` (page 95) on *bundle*, passing it the specified *key*, *value*, and *tableName*.

Discussion
You can specify Unicode characters in *key* using \\Uxxxx—see the -u option for for the genstrings utility.

If you use genstrings to parse your code for localizable strings, you can use this method to specify an initial value that is different from key.

For more information, see NSBundle.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSBundle.h

NSLocationInRange

Returns a Boolean value that indicates whether a specified position is in a given range.

```
BOOL NSLocationInRange (
   NSUInteger loc,
   NSRange range
);
```

Return Value
YES if *index* lies within *aRange*—that is, if it's greater than or equal to aRange.location and less than aRange.location plus aRange.length.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRange.h

NSLog

Logs error an message to stderr.

```
void NSLog (
   NSString *format,
   ...
);
```

Discussion
Simply calls NSLogv (page 1365), passing it a variable number of arguments.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSLogv (page 1365)

Declared In
NSObjCRuntime.h

NSLogPageSize

Returns the binary log of the page size.

```
NSUInteger NSLogPageSize (
   void
);
```

Return Value

The binary log of the page size.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSRoundDownToMultipleOfPageSize (page 1372)
NSRoundUpToMultipleOfPageSize (page 1372)
NSPageSize (page 1367)

Declared In

NSZone.h

NSLogv

Logs an error message to stderr.

```
void NSLogv (
   NSString *format,
   va_list args
);
```

Discussion

Logs an error message. The message consists of a timestamp and the process ID prefixed to the string you pass in. You compose this string with a format string, *format*, and one or more arguments to be inserted into the string. The format specification allowed by these functions is that which is understood by NSString's formatting capabilities (which is not necessarily the set of format escapes and flags understood by printf). The supported format specifiers are described in String Format Specifiers. A final hard return is added to the error message if one is not present in the format.

In general, you should use the NSLog (page 1364) function instead of calling this function directly. If you do use this function directly, you must have prepared the variable argument list in the *args* argument by calling the standard C macro va_start. Upon completion, you must similarly call the standard C macro va_end for this list.

NSLogv writes the log to STDERR_FILENO if the file descriptor is open. If that write attempt fails, the message is sent to the syslog subsystem, if it exists on a platform, with the LOG_USER facility (or default facility if LOG_USER does not exist), with priority LOG_ERR (or similar). If both of these attempts to write the message fail, the message is discarded.

Output from NSLogv is serialized, in that only one thread in a process can be doing the writing/logging described above at a time. All attempts at writing/logging a message complete before the next thread can begin its attempts.

The effects of `NSLogv` are not serialized with subsystems other than those discussed above (such as the standard I/O package) and do not produce side effects on those subsystems (such as causing buffered output to be flushed, which may be undesirable).

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSLog` (page 1364)

Declared In
`NSObjCRuntime.h`

NSMakeCollectable

Makes a newly allocated Core Foundation object eligible for collection.

```
NS_INLINE id NSMakeCollectable(CFTypeRef cf) {
    return cf ? (id)CFMakeCollectable(cf) : nil;
}
```

Discussion
This function is a wrapper for `CFMakeCollectable`, but its return type is `id`—avoiding the need for casting when using Cocoa objects.

This function may be useful when returning Core Foundation objects in code that must support both garbage-collected and non-garbage-collected environments, as illustrated in the following example.

```
- (CFDateRef)foo {
    CFDateRef aCFDate;
    // ...
    return [NSMakeCollectable(aCFDate) autorelease];
}
```

CFTypeRef style objects are garbage collected, yet only sometime after the last `CFRelease` is performed. Particularly for fully-bridged CFTypeRef objects such as CFStrings and collections (such as CFDictionary), you must call either `CFMakeCollectable` or the more type safe `NSMakeCollectable`, preferably right upon allocation.

Declared In
`NSZone.h`

NSMakeRange

Creates a new NSRange from the specified values.

```
NSRange NSMakeRange (
   NSUInteger loc,
   NSUInteger len
);
```

Return Value
An `NSRange` with location *`location`* and length *`length`*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRange.h

NSMaxRange

Returns the number 1 greater than the maximum value within the range.

```
NSUInteger NSMaxRange (
   NSRange range
);
```

Return Value
`range.location` + `range.length`—in other words, the number 1 greater than the maximum value within the range.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRange.h

NSOpenStepRootDirectory

Returns the root directory of the user's system.

```
NSString * NSOpenStepRootDirectory (
   void
);
```

Return Value
A string identifying the root directory of the user's system.

Discussion
For more information on file system utilities, see Low-Level File Management Programming Topics.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSHomeDirectory` (page 1360)
`NSHomeDirectoryForUser` (page 1360)

Declared In
NSPathUtilities.h

NSPageSize

Returns the number of bytes in a page.

```
NSUInteger NSPageSize (
   void
);
```

Return Value
The number of bytes in a page.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSRoundDownToMultipleOfPageSize (page 1372)
NSRoundUpToMultipleOfPageSize (page 1372)
NSLogPageSize (page 1365)

Declared In
NSZone.h

NSParameterAssert

Validates the specified parameter.

```
NSParameterAssert(condition)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

This macro validates a parameter for an Objective-C method. Simply provide the parameter as the *`condition`* argument. The macro evaluates the parameter and, if it is false, it logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All assertion macros return void.

See Also
NSLog (page 1364)
NSLogv (page 1365)
NSAssert (page 1336)
NSCAssert (page 1341)
NSCParameterAssert (page 1348)

NSPointFromCGPoint

Returns an `NSPoint` typecast from a `CGPoint`.

```
NSPoint NSPointFromCGPoint(CGPoint cgpoint) {
    return (*(NSPoint *)&(cgpoint));
}
```

Return Value
An `NSPoint` typecast from a `CGPoint`.

Declared In
`NSGeometry.h`

See Also
`NSPointToCGPoint` (page 1369)
`NSRectFromCGRect` (page 1370)
`NSSizeFromCGSize` (page 1375)

NSPointToCGPoint

Returns a `CGPoint` typecast from an `NSPoint`.

```
CGPoint NSPointToCGPoint(NSPoint nspoint) {
    return (*(CGPoint *)&(nspoint));
}
```

Return Value
A `CGPoint` typecast from an `NSPoint`.

Declared In
`NSGeometry.h`

See Also
`NSPointFromCGPoint` (page 1368)
`NSRectToCGRect` (page 1371)
`NSSizeToCGSize` (page 1375)

NSProtocolFromString

Returns a the protocol with a given name.

```
Protocol *NSProtocolFromString (
    NSString *namestr
);
```

Parameters

`namestr`
The name of a protocol.

Return Value
The protocol object named by *`namestr`*, or `nil` if no protocol by that name is currently loaded. If *`namestr`* is `nil`, returns `nil`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSStringFromProtocol` (page 1376)

NSClassFromString (page 1345)
NSSelectorFromString (page 1373)

Declared In
NSObjCRuntime.h

NSRangeFromString

Returns a range from a text-based representation.

```
NSRange NSRangeFromString (
   NSString *aString
);
```

Discussion
Scans *aString* for two integers which are used as the location and length values, in that order, to create an NSRange struct. If *aString* only contains a single integer, it is used as the location value. If *aString* does not contain any integers, this function returns an NSRange struct whose location and length values are both 0.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSStringFromRange (page 1377)

Declared In
NSRange.h

NSRealMemoryAvailable

Returns information about the user's system.

```
NSUInteger NSRealMemoryAvailable (
   void
);
```

Return Value
The number of bytes available in RAM.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSZone.h

NSRectFromCGRect

Returns an NSRect typecast from a CGRect.

```
NSRect NSRectFromCGRect(CGRect cgrect) {
    return (*(NSRect *)&(cgrect));
}
```

Return Value
An `NSRect` typecast from a `CGRect`.

Declared In
`NSGeometry.h`

See Also
`NSRectToCGRect` (page 1371)
`NSPointFromCGPoint` (page 1368)
`NSSizeFromCGSize` (page 1375)

NSRectToCGRect

Returns a `CGRect` typecast from an `NSRect`.

```
CGRect NSRectToCGRect(NSRect nsrect) {
    return (*(CGRect *)&(nsrect));
}
```

Return Value
A `CGRect` typecast from an `NSRect`.

Declared In
`NSGeometry.h`

See Also
`NSRectFromCGRect` (page 1370)
`NSPointToCGPoint` (page 1369)
`NSSizeToCGSize` (page 1375)

NSRecycleZone

Frees memory in a zone.

```
void NSRecycleZone (
   NSZone *zone
);
```

Discussion
Frees *`zone`* after adding any of its pointers still in use to the default zone. (This strategy prevents retained objects from being inadvertently destroyed.)

Returns `void`.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSCreateZone` (page 1349)
`NSZoneMalloc` (page 1395)

Declared In
NSZone.h

NSRoundDownToMultipleOfPageSize

Returns the specified number of bytes rounded down to a multiple of the page size.

```
NSUInteger NSRoundDownToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value
In bytes, the multiple of the page size that is closest to, but not greater than, *byteCount* (that is, the number of bytes rounded down to a multiple of the page size).

Availability
Available in iPhone OS 2.0 and later.

See Also
NSPageSize (page 1367)
NSLogPageSize (page 1365)
NSRoundUpToMultipleOfPageSize (page 1372)

Declared In
NSZone.h

NSRoundUpToMultipleOfPageSize

Returns the specified number of bytes rounded up to a multiple of the page size.

```
NSUInteger NSRoundUpToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value
In bytes, the multiple of the page size that is closest to, but not less than, *byteCount* (that is, the number of bytes rounded up to a multiple of the page size).

Availability
Available in iPhone OS 2.0 and later.

See Also
NSPageSize (page 1367)
NSLogPageSize (page 1365)
NSRoundDownToMultipleOfPageSize (page 1372)

Declared In
NSZone.h

NSSearchPathForDirectoriesInDomains

Creates a list of directory search paths.

```
NSArray * NSSearchPathForDirectoriesInDomains (
   NSSearchPathDirectory directory,
   NSSearchPathDomainMask domainMask,
   BOOL expandTilde
);
```

Discussion

Creates a list of path strings for the specified directories in the specified domains. The list is in the order in which you should search the directories. If *expandTilde* is YES, tildes are expanded as described in stringByExpandingTildeInPath (page 1031).

For more information on file system utilities, see Locating Directories on the System.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSPathUtilities.h

NSSelectorFromString

Returns the selector with a given name.

```
SEL NSSelectorFromString (
   NSString *aSelectorName
);
```

Parameters

aSelectorName

A string of any length, with any characters, that represents the name of a selector.

Return Value

The selector named by *aSelectorName*. If *aSelectorName* is nil, or cannot be converted to UTF-8 (this should be only due to insufficient memory), returns (SEL)0.

Discussion

To make a selector, NSSelectorFromString passes a UTF-8 encoded character representation of *aSelectorName* to sel_registerName and returns the value returned by that function. Note, therefore, that if the selector does not exist it is registered and the newly-registered selector is returned.

Recall that a colon (":") is part of a method name; setHeight is not the same as setHeight:. For more about methods names, see The Language in *The Objective-C 2.0 Programming Language*.

Availability

Available in iPhone OS 2.0 and later.

See Also

NSStringFromSelector (page 1377)
NSProtocolFromString (page 1369)
NSClassFromString (page 1345)

Declared In

NSObjCRuntime.h

NSSetUncaughtExceptionHandler

Changes the top-level error handler.

```
void NSSetUncaughtExceptionHandler (
   NSUncaughtExceptionHandler *
);
```

Discussion

Sets the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSGetUncaughtExceptionHandler` (page 1359)

`reportException:` (`NSApplication`)

Declared In

`NSException.h`

NSSetZoneName

Sets the name of the specified zone.

```
void NSSetZoneName (
   NSZone *zone,
   NSString *name
);
```

Discussion

Sets the name of *`zone`* to *`name`*, which can aid in debugging.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSZoneName` (page 1395)

Declared In

`NSZone.h`

NSShouldRetainWithZone

Indicates whether an object should be retained.

```
BOOL NSShouldRetainWithZone (
   id anObject,
   NSZone *requestedZone
);
```

Parameters

anObject
An object.

requestedZone
A memory zone.

Return Value

Returns YES if *requestedZone* is NULL, the default zone, or the zone in which *anObject* was allocated; otherwise NO.

Discussion

This function is typically called from inside an NSObject's copyWithZone: (page 787), when deciding whether to retain *anObject* as opposed to making a copy of it.

Availability

Available in iPhone OS 2.0 and later.

Declared In

NSObject.h

NSSizeFromCGSize

Returns an NSSize typecast from a CGSize.

```
NSSize NSSizeFromCGSize(CGSize cgsize) {
    return (*(NSSize *)&(cgsize));
}
```

Return Value

An NSSize typecast from a CGSize.

Declared In

NSGeometry.h

See Also

NSSizeToCGSize (page 1375)
NSPointFromCGPoint (page 1368)
NSRectFromCGRect (page 1370)

NSSizeToCGSize

Returns a CGSize typecast from an NSSize.

```
CGSize NSSizeToCGSize(NSSize nssize) {
    return (*(CGSize *)&(nssize));
}
```

Return Value

A CGSize typecast from an NSSize.

Declared In
NSGeometry.h

See Also
NSSizeFromCGSize (page 1375)
NSPointToCGPoint (page 1369)
NSRectToCGRect (page 1371)

NSStringFromClass

Returns the name of a class as a string.

```
NSString * NSStringFromClass (
   Class aClass
);
```

Parameters

aClass
A class.

Return Value
A string containing the name of *aClass*. If *aClass* is nil, returns nil.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSClassFromString (page 1345)
NSStringFromProtocol (page 1376)
NSStringFromSelector (page 1377)

Declared In
NSObjCRuntime.h

NSStringFromProtocol

Returns the name of a protocol as a string.

```
NSString * NSStringFromProtocol (
    Protocol *proto
);
```

Parameters

proto
A protocol.

Return Value
A string containing the name of *proto*.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSProtocolFromString (page 1369)

NSStringFromClass (page 1376)
NSStringFromSelector (page 1377)

Declared In
NSObjCRuntime.h

NSStringFromRange

Returns a string representation of a range.

```
NSString * NSStringFromRange (
   NSRange range
);
```

Return Value
A string of the form "{*a*, *b*}", where *a* and *b* are non-negative integers representing *aRange*.

Availability
Available in iPhone OS 2.0 and later.

Declared In
NSRange.h

NSStringFromSelector

Returns a string representation of a given selector.

```
NSString *NSStringFromSelector (
    SEL *aSelector
);
```

Parameters
aSelector
A selector.

Return Value
A string representation of *aSelector*.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSelectorFromString (page 1373)
NSStringFromProtocol (page 1376)
NSStringFromClass (page 1376)

Declared In
NSObjCRuntime.h

NSSwapBigDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapBigDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapDouble` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostDoubleToBig` (page 1381)
`NSSwapLittleDoubleToHost` (page 1387)

Declared In
`NSByteOrder.h`

NSSwapBigFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapBigFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapFloat` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostFloatToBig` (page 1382)
`NSSwapLittleFloatToHost` (page 1388)

Declared In
`NSByteOrder.h`

NSSwapBigIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapBigIntToHost (
   unsigned int x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapInt` (page 1387) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapHostIntToBig (page 1383)
NSSwapLittleIntToHost (page 1388)

Declared In
NSByteOrder.h

NSSwapBigLongLongToHost

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapBigLongLongToHost (
   unsigned long long x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapLongLong (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapHostLongLongToBig (page 1384)
NSSwapLittleLongLongToHost (page 1389)

Declared In
NSByteOrder.h

NSSwapBigLongToHost

A utility for swapping the bytes of a number.

```
unsigned long NSSwapBigLongToHost (
   unsigned long x
);
```

Discussion
Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls NSSwapLong (page 1390) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapHostLongToBig (page 1385)
NSSwapLittleLongToHost (page 1389)

Declared In
NSByteOrder.h

NSSwapBigShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapBigShortToHost (
   unsigned short x
);
```

Discussion

Converts the big-endian value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapShort` (page 1391) to perform the swap.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSSwapHostShortToBig` (page 1386)
`NSSwapLittleShortToHost` (page 1390)

Declared In

`NSByteOrder.h`

NSSwapDouble

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapDouble (
   NSSwappedDouble x
);
```

Discussion

Swaps the bytes of *x* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *x* are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability

Available in iPhone OS 2.0 and later.

See Also

`NSSwapLongLong` (page 1391)
`NSSwapFloat` (page 1380)

Declared In

`NSByteOrder.h`

NSSwapFloat

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapFloat (
   NSSwappedFloat x
);
```

Discussion
Swaps the bytes of *x* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *x* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLong` (page 1390)
`NSSwapDouble` (page 1380)

Declared In
`NSByteOrder.h`

NSSwapHostDoubleToBig

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToBig (
   double x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigDoubleToHost` (page 1377)
`NSSwapHostDoubleToLittle` (page 1381)

Declared In
`NSByteOrder.h`

NSSwapHostDoubleToLittle

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToLittle (
   double x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleDoubleToHost` (page 1387)
`NSSwapHostDoubleToBig` (page 1381)

Declared In
`NSByteOrder.h`

NSSwapHostFloatToBig

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToBig (
   float x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigFloatToHost` (page 1378)
`NSSwapHostFloatToLittle` (page 1382)

Declared In
`NSByteOrder.h`

NSSwapHostFloatToLittle

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToLittle (
   float x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleFloatToHost` (page 1388)
`NSSwapHostFloatToBig` (page 1382)

Declared In
`NSByteOrder.h`

NSSwapHostIntToBig

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToBig (
   unsigned int x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 1387) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigIntToHost` (page 1378)
`NSSwapHostIntToLittle` (page 1383)

Declared In
`NSByteOrder.h`

NSSwapHostIntToLittle

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToLittle (
   unsigned int x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 1387) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleIntToHost` (page 1388)
`NSSwapHostIntToBig` (page 1383)

Declared In
`NSByteOrder.h`

NSSwapHostLongLongToBig

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToBig (
   unsigned long long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigLongLongToHost` (page 1379)
`NSSwapHostLongLongToLittle` (page 1384)

Declared In
`NSByteOrder.h`

NSSwapHostLongLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToLittle (
   unsigned long long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleLongLongToHost` (page 1389)
`NSSwapHostLongLongToBig` (page 1384)

Declared In
`NSByteOrder.h`

NSSwapHostLongToBig

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToBig (
   unsigned long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLong` (page 1390) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigLongToHost` (page 1379)
`NSSwapHostLongToLittle` (page 1385)

Declared In
`NSByteOrder.h`

NSSwapHostLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToLittle (
   unsigned long x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLong` (page 1390) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleLongToHost` (page 1389)
`NSSwapHostLongToBig` (page 1385)

Declared In
`NSByteOrder.h`

NSSwapHostShortToBig

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToBig (
   unsigned short x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapShort` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapBigShortToHost` (page 1380)
`NSSwapHostShortToLittle` (page 1386)

Declared In
`NSByteOrder.h`

NSSwapHostShortToLittle

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToLittle (
   unsigned short x
);
```

Discussion
Converts the value in *x*, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapShort` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapLittleShortToHost` (page 1390)
`NSSwapHostShortToBig` (page 1386)

Declared In
`NSByteOrder.h`

NSSwapInt

A utility for swapping the bytes of a number.

```
unsigned int NSSwapInt (
   unsigned int inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapShort` (page 1391)
`NSSwapLong` (page 1390)
`NSSwapLongLong` (page 1391)

Declared In
`NSByteOrder.h`

NSSwapLittleDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapLittleDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapDouble` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostDoubleToLittle` (page 1381)
`NSSwapBigDoubleToHost` (page 1377)
`NSConvertSwappedDoubleToHost` (page 1346)

Declared In
`NSByteOrder.h`

NSSwapLittleFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapLittleFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapFloat` (page 1380) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostFloatToLittle` (page 1382)
`NSSwapBigFloatToHost` (page 1378)
`NSConvertSwappedFloatToHost` (page 1347)

Declared In
`NSByteOrder.h`

NSSwapLittleIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapLittleIntToHost (
   unsigned int x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 1387) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostIntToLittle` (page 1383)
`NSSwapBigIntToHost` (page 1378)

Declared In
`NSByteOrder.h`

NSSwapLittleLongLongToHost
A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLittleLongLongToHost (
   unsigned long long x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostLongLongToLittle` (page 1384)
`NSSwapBigLongLongToHost` (page 1379)

Declared In
`NSByteOrder.h`

NSSwapLittleLongToHost
A utility for swapping the bytes of a number.

```
unsigned long NSSwapLittleLongToHost (
   unsigned long x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls `NSSwapLong` (page 1390) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostLongToLittle` (page 1385)
`NSSwapBigLongToHost` (page 1379)
`NSSwapLong` (page 1390)

Declared In
`NSByteOrder.h`

NSSwapLittleShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapLittleShortToHost (
   unsigned short x
);
```

Discussion
Converts the little-endian formatted value in *x* to the current endian format and returns the resulting value. If it is necessary to swap the bytes of *x*, this function calls `NSSwapShort` (page 1391) to perform the swap.

Availability
Available in iPhone OS 2.0 and later.

See Also
`NSSwapHostShortToLittle` (page 1386)
`NSSwapBigShortToHost` (page 1380)

Declared In
`NSByteOrder.h`

NSSwapLong

A utility for swapping the bytes of a number.

```
unsigned long NSSwapLong (
   unsigned long inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapLongLong (page 1391)
NSSwapInt (page 1387)
NSSwapFloat (page 1380)

Declared In
NSByteOrder.h

NSSwapLongLong

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLongLong (
   unsigned long long inv
);
```

Discussion
Swaps the bytes of *inv* and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of *inv* are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapLong (page 1390)
NSSwapDouble (page 1380)

Declared In
NSByteOrder.h

NSSwapShort

A utility for swapping the bytes of a number.

```
unsigned short NSSwapShort (
   unsigned short inv
);
```

Discussion
Swaps the low-order and high-order bytes of *inv* and returns the resulting value.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSwapInt (page 1387)
NSSwapLong (page 1390)

Declared In
NSByteOrder.h

NSTemporaryDirectory

Returns the path of the temporary directory for the current user.

```
NSString * NSTemporaryDirectory (
   void
);
```

Return Value
A string containing the path of the temporary directory for the current user. If no such directory is currently available, returns nil.

Discussion
For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Files put in the temporary directory may be moved to the Trash in a "Recovered Files" directory when the user's system is restarted. You should therefore ensure that you delete temporary files before your application terminates.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSearchPathForDirectoriesInDomains (page 1372)
NSHomeDirectory (page 1360)

Declared In
NSPathUtilities.h

NSUnionRange

Returns the intersection of the specified ranges.

```
NSRange NSUnionRange (
   NSRange range1,
   NSRange range2
);
```

Return Value
A range covering all indices in and between *range1* and *range2*. If one range is completely contained in the other, the returned range is equal to the larger range.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSIntersectionRange (page 1361)

Declared In
NSRange.h

NSUserName
Returns the logon name of the current user.

```
NSString * NSUserName (
   void
);
```

Return Value
The logon name of the current user.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSFullUserName (page 1358)
NSHomeDirectory (page 1360)
NSHomeDirectoryForUser (page 1360)

Declared In
NSPathUtilities.h

NSZoneCalloc
Allocates memory in a zone.

```
void * NSZoneCalloc (
   NSZone *zone,
   NSUInteger numElems,
   NSUInteger byteSize
);
```

Discussion
Allocates enough memory from *zone* for *numElems* elements, each with a size *numBytes* bytes, and returns a pointer to the allocated memory. The memory is initialized with zeros. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSDefaultMallocZone (page 1357)
NSRecycleZone (page 1371)
NSZoneFree (page 1394)
NSZoneMalloc (page 1395)
NSZoneRealloc (page 1396)

Declared In
NSZone.h

NSZoneFree

Deallocates a block of memory in the specified zone.

```
void NSZoneFree (
   NSZone *zone,
   void *ptr
);
```

Discussion
Returns memory to the *zone* from which it was allocated. The standard C function `free` does the same, but spends time finding which zone the memory belongs to.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSRecycleZone (page 1371)
NSZoneMalloc (page 1395)
NSZoneCalloc (page 1393)
NSZoneRealloc (page 1396)

Declared In
NSZone.h

NSZoneFromPointer

Gets the zone for a given block of memory.

```
NSZone * NSZoneFromPointer (
   void *ptr
);
```

Return Value
The zone for the block of memory indicated by *pointer*, or `NULL` if the block was not allocated from a zone.

Discussion
pointer must be one that was returned by a prior call to an allocation function.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSZoneCalloc (page 1393)
NSZoneMalloc (page 1395)
NSZoneRealloc (page 1396)

Declared In
NSZone.h

NSZoneMalloc

Allocates memory in a zone.

```
void * NSZoneMalloc (
   NSZone *zone,
   NSUInteger size
);
```

Discussion
Allocates *size* bytes in *zone* and returns a pointer to the allocated memory. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSDefaultMallocZone (page 1357)
NSRecycleZone (page 1371)
NSZoneFree (page 1394)
NSZoneCalloc (page 1393)
NSZoneRealloc (page 1396)

Declared In
NSZone.h

NSZoneName

Returns the name of the specified zone.

```
NSString * NSZoneName (
   NSZone *zone
);
```

Return Value
A string containing the name associated with *zone*. If *zone* is nil, the default zone is used. If no name is associated with *zone*, the returned string is empty.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSSetZoneName (page 1374)

Declared In
NSZone.h

NSZoneRealloc

Allocates memory in a zone.

```
void * NSZoneRealloc (
   NSZone *zone,
   void *ptr,
   NSUInteger size
);
```

Discussion
Changes the size of the block of memory pointed to by *ptr* to *size* bytes. It may allocate new memory to replace the old, in which case it moves the contents of the old memory block to the new block, up to a maximum of *size* bytes. *ptr* may be NULL. This function returns NULL if it was unable to allocate the requested memory.

Availability
Available in iPhone OS 2.0 and later.

See Also
NSDefaultMallocZone (page 1357)
NSRecycleZone (page 1371)
NSZoneFree (page 1394)
NSZoneCalloc (page 1393)
NSZoneMalloc (page 1395)

Declared In
NSZone.h

NS_DURING

Marks the start of the exception-handling domain.

```
NS_DURING
```

Discussion
The NS_DURING macro marks the start of the exception-handling domain for a section of code. (The NS_HANDLER (page 1397)macro marks the end of the domain.) Within the exception-handling domain you can raise an exception, giving the local exception handler (or lower exception handlers) a chance to handle it.

NS_ENDHANDLER

Marks the end of the local event handler.

```
NS_ENDHANDLER
```

Discussion
The NS_ENDHANDLER marks the end of a section of code that is a local exception handler. (The NS_HANDLER (page 1397)macros marks the beginning of this section.) If an exception is raised in the exception handling domain marked off by the NS_DURING (page 1396) and NS_HANDLER (page 1397), the local exception handler (if specified) is given a chance to handle the exception.

NS_HANDLER

Marks the end of the exception-handling domain and the start of the local exception handler.

```
NS_HANDLER
```

Discussion
The NS_HANDLER macro marks end of a section of code that is an exception-handling domain while at the same time marking the beginning of a section of code that is a local exception handler for that domain. (The NS_DURING (page 1396) macro marks the beginning of the exception-handling domain; the NS_ENDHANDLER (page 1396) marks the end of the local exception handler.) If an exception is raised in the exception-handling domain, the local exception handler is first given the chance to handle the exception before lower-level handlers are given a chance.

NS_VALUERETURN

Permits program control to exit from an exception-handling domain with a value of a specified type.

```
NS_VALUERETURN(val, type)
```

Parameters

val

A value to preserve beyond the exception-handling domain.

type

The type of the value specified in val.

Discussion
The NS_VALUERETURN macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the NS_DURING (page 1396) and NS_HANDLER (page 1397) macros that might raise an exception. The specified value (of the specified type) is returned to the caller. The standard return statement does not work as expected in the exception-handling domain.

NS_VOIDRETURN

Permits program control to exit from an exception-handling domain.

```
NS_VOIDRETURN
```

Discussion
The NS_VOIDRETURN macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the NS_DURING (page 1396) and NS_HANDLER (page 1397) macros that might raise an exception. The standard return statement does not work as expected in the exception-handling domain.

Data Types

PART IV

Data Types

Foundation Data Types Reference

Framework: Foundation/Foundation.h

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

This document describes the data types and constants found in the Foundation framework.

Data Types

NSByteOrder

These constants specify an endian format.

```
enum _NSByteOrder {
    NS_UnknownByteOrder = CFByteOrderUnknown,
    NS_LittleEndian = CFByteOrderLittleEndian,
    NS_BigEndian = CFByteOrderBigEndian
};
```

Constants

`NS_UnknownByteOrder`
The byte order is unknown.
Available in iPhone OS 2.0 and later.
Declared in `NSByteOrder.h`

`NS_LittleEndian`
The byte order is little endian.
Available in iPhone OS 2.0 and later.
Declared in `NSByteOrder.h`

`NS_BigEndian`
The byte order is big endian.
Available in iPhone OS 2.0 and later.
Declared in `NSByteOrder.h`

Discussion

These constants are returned by `NSHostByteOrder` (page 1361).

Declared In

`NSByteOrder.h`

NSComparisonResult

These constants are used to indicate how items in a request are ordered.

```
typedef enum _NSComparisonResult {
    NSOrderedAscending = -1,
    NSOrderedSame,
    NSOrderedDescending
} NSComparisonResult;
```

Constants

`NSOrderedAscending`
The left operand is smaller than the right operand.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSOrderedSame`
The two operands are equal.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSOrderedDescending`
The left operand is greater than the right operand.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

Discussion
These constants are used to indicate how items in a request are ordered, from the first one given in a method invocation or function call to the last (that is, left to right in code).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSObjCRuntime.h`

NSDecimal

Used to describe a decimal number.

```
typedef struct {
    signed int _exponent:8;
    unsigned int _length:4;
    unsigned int _isNegative:1;
    unsigned int _isCompact:1;
    unsigned int _reserved:18;
    unsigned short _mantissa[NSDecimalMaxSize];
} NSDecimal;
```

Discussion
The fields of `NSDecimal` are private.

Used by the functions described in "Decimals" (page 1330).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDecimal.h`

NSHashTableOptions

Specifies a bitfield used to configure the behavior of elements in an instance of `NSHashTable`.

```
typedef NSUInteger NSHashTableOptions
```

Declared In
`NSHashTable.h`

NSMapTableOptions

Specifies a bitfield used to configure the behavior of elements in an instance of `NSMapTable`.

```
typedef NSUInteger NSMapTableOptions
```

Declared In
`NSMapTable.h`

NSRange

A structure used to describe a portion of a series—such as characters in a string or objects in an `NSArray` object.

```
typedef struct _NSRange {
    NSUInteger location;
    NSUInteger length;
} NSRange;
```

Fields

`location`

The start index (`0` is the first, as in C arrays).

`length`

The number of items in the range (can be `0`).

Discussion

Foundation functions that operate on ranges include the following:

- `NSEqualRanges` (page 1358)
- `NSIntersectionRange` (page 1361)
- `NSLocationInRange` (page 1364)
- `NSMakeRange` (page 1366)
- `NSMaxRange` (page 1367)
- `NSRangeFromString` (page 1370)
- `NSStringFromRange` (page 1377)
- `NSUnionRange` (page 1392)

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRange.h`

NSRangePointer

Type indicating a parameter is a pointer to an `NSRange` structure.

```
typedef NSRange *NSRangePointer;
```

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSRange.h`

NSSearchPathDirectory

These constants specify the location of a variety of directories.

```
typedef enum {
    NSApplicationDirectory = 1,
    NSDemoApplicationDirectory,
    NSDeveloperApplicationDirectory,
    NSAdminApplicationDirectory,
    NSLibraryDirectory,
    NSDeveloperDirectory,
    NSUserDirectory,
    NSDocumentationDirectory,
    NSDocumentDirectory,
    NSCoreServiceDirectory,
    NSDesktopDirectory = 12,
    NSCachesDirectory = 13,
    NSApplicationSupportDirectory = 14,
    NSDownloadsDirectory = 15,
    NSAllApplicationsDirectory = 100,
    NSAllLibrariesDirectory = 101
} NSSearchPathDirectory;
```

Constants

`NSApplicationDirectory`
Supported applications (`/Applications`).
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSDemoApplicationDirectory`
Unsupported applications and demonstration versions.
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSDeveloperApplicationDirectory`
Developer applications (`/Developer/Applications`).
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSAdminApplicationDirectory`
System and network administration applications.
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSLibraryDirectory`
Various user-visible documentation, support, and configuration files (`/Library`).
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSDeveloperDirectory`
Developer resources (`/Developer`).
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSUserDirectory`
User home directories (`/Users`).
Available in iPhone OS 2.0 and later.
Declared in `NSPathUtilities.h`

`NSDocumentationDirectory`
Documentation.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSDocumentDirectory`
Document directory.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSCoreServiceDirectory`
Location of core services (`System/Library/CoreServices`).

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSDesktopDirectory`
Location of user's desktop directory.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSCachesDirectory`
Location of discardable cache files (`Library/Caches`).

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSApplicationSupportDirectory`
Location of application support files (`Library/Application Support`).

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSDownloadsDirectory`
Location of the user's downloads directory.

The `NSDownloadsDirectory` flag will only produce a path only when the `NSUserDomainMask` is provided.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSAllApplicationsDirectory`
All directories where applications can occur.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSAllLibrariesDirectory`
All directories where resources can occur.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSPathUtilities.h`

NSSearchPathDomainMask

Search path domain constants specifying base locations for the NSSearchPathDirectory (page 1404) type.

```
typedef enum {
    NSUserDomainMask = 1,
    NSLocalDomainMask = 2,
    NSNetworkDomainMask = 4,
    NSSystemDomainMask = 8,
    NSAllDomainsMask = 0xffff,
} NSSearchPathDomainMask;
```

Constants

`NSUserDomainMask`

The user's home directory—the place to install user's personal items (~).

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSLocalDomainMask`

Local to the current machine—the place to install items available to everyone on this machine.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSNetworkDomainMask`

Publicly available location in the local area network—the place to install items available on the network (`/Network`).

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSSystemDomainMask`

Provided by Apple — can't be modified (`/System`) .

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

`NSAllDomainsMask`

All domains.

Includes all of the above and future items.

Available in iPhone OS 2.0 and later.

Declared in `NSPathUtilities.h`

Availability

Available in iPhone OS 2.0 and later.

Declared In

`NSPathUtilities.h`

NSStringEncoding

Type representing string-encoding values.

```
typedef NSUInteger NSStringEncoding;
```

Discussion
See String Encodings (page 1045) for a list of values.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSString.h`

NSSwappedDouble

Opaque structure containing endian-independent `double` value.

```
typedef struct {
    unsigned long long v;
} NSSwappedDouble;
```

Discussion
The fields of an `NSSwappedDouble` are private.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSByteOrder.h`

NSSwappedFloat

Opaque type containing an endian-independent `float` value.

```
typedef struct {
    unsigned long v;
} NSSwappedFloat;
```

Discussion
The fields of an `NSSwappedFloat` are private.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSByteOrder.h`

NSTimeInterval

Used to specify a time interval, in seconds.

```
typedef double NSTimeInterval;
```

Discussion
`NSTimeInterval` is always specified in seconds; it yields sub-millisecond precision over a range of 10,000 years.

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSDate.h`

NSUncaughtExceptionHandler

Used for the function handling exceptions outside of an exception-handling domain.

```
typedef volatile void NSUncaughtExceptionHandler(NSException *exception);
```

Discussion
You can set exception handlers using `NSSetUncaughtExceptionHandler` (page 1374).

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSException.h`

NSZone

Used to identify and manage memory zones.

```
typedef struct _NSZone NSZone;
```

Availability
Available in iPhone OS 2.0 and later.

Declared In
`NSZone.h`

Constants

PART V

Constants

Foundation Constants Reference

Framework: Foundation/Foundation.h

Overview

Important: This is a Mac OS X document for an API or technology that is shared between Mac OS X and iPhone OS. Although this document has been reviewed for technical accuracy on Mac OS X, it has not been reviewed for accuracy on iPhone OS and may contain errors or omissions. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future seeds of the API or technology. For information about updates to this and other developer documentation, view the New & Updated sidebars in subsequent documentation seeds.

This document defines constants in the Foundation framework that are not associated with a particular class.

Constants

Enumerations

NSError Codes

`NSError` codes in the Cocoa error domain.

```
enum {
    NSFileNoSuchFileError = 4,
    NSFileLockingError = 255,
    NSFileReadUnknownError = 256,
    NSFileReadNoPermissionError = 257,
    NSFileReadInvalidFileNameError = 258,
    NSFileReadCorruptFileError = 259,
    NSFileReadNoSuchFileError = 260,
    NSFileReadInapplicableStringEncodingError = 261,
    NSFileReadUnsupportedSchemeError = 262,
    NSFileWriteUnknownError = 512,
    NSFileWriteNoPermissionError = 513,
    NSFileWriteInvalidFileNameError = 514,
    NSFileWriteInapplicableStringEncodingError = 517,
    NSFileWriteUnsupportedSchemeError = 518,
    NSFileWriteOutOfSpaceError = 640,
    NSKeyValueValidationError = 1024,
    NSFormattingError = 2048,
    NSUserCancelledError = 3072,

    NSFileErrorMinimum = 0,
    NSFileErrorMaximum = 1023,
    NSValidationErrorMinimum = 1024,
    NSValidationErrorMaximum = 2047,
    NSFormattingErrorMinimum = 2048,
    NSFormattingErrorMaximum = 2559,

    NSExecutableErrorMinimum = 3584,
    NSExecutableNotLoadableError = 3584,
    NSExecutableArchitectureMismatchError = 3585,
    NSExecutableRuntimeMismatchError = 3586,
    NSExecutableLoadError = 3587,
    NSExecutableLinkError = 3588,
    NSExecutableErrorMaximum = 3839
}
```

Constants

`NSFileNoSuchFileError`

File-system operation attempted on non-existent file

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileLockingError`

Failure to get a lock on file

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadUnknownError`

Read error, reason unknown

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadNoPermissionError`

Read error because of a permission problem

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadInvalidFileNameError`
Read error because of an invalid file name

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadCorruptFileError`
Read error because of a corrupted file, bad format, or similar reason

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadNoSuchFileError`
Read error because no such file was found

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadInapplicableStringEncodingError`
Read error because the string encoding was not applicable.

Access the bad encoding from the `userInfo` dictionary using the `NSStringEncodingErrorKey` key.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileReadUnsupportedSchemeError`
Read error because the specified URL scheme is unsupported

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteUnknownError`
Write error, reason unknown

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteNoPermissionError`
Write error because of a permission problem

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteInvalidFileNameError`
Write error because of an invalid file name

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteInapplicableStringEncodingError`
Write error because the string encoding was not applicable.

Access the bad encoding from the `userInfo` dictionary using the `NSStringEncodingErrorKey` key.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteUnsupportedSchemeError`
Write error because the specified URL scheme is unsupported

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileWriteOutOfSpaceError`
Write error because of a lack of disk space

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSKeyValueValidationError`
Key-value coding validation error

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFormattingError`
Formatting error (related to display of data)

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSUserCancelledError`
The user cancelled the operation (for example, by pressing Command-period).

This code is for errors that do not require a dialog displayed and might be candidates for special-casing.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileErrorMinimum`
Marks the start of the range of error codes reserved for file errors

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFileErrorMaximum`
Marks the end of the range of error codes reserved for file errors

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSValidationErrorMinimum`
Marks the start of the range of error codes reserved for validation errors.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSValidationErrorMaximum`
Marks the start and end of the range of error codes reserved for validation errors.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFormattingErrorMinimum`
Marks the start of the range of error codes reserved for formatting errors.

Available in iPhone OS 2.0 and later.

Declared in `FoundationErrors.h`

`NSFormattingErrorMaximum`
Marks end of the range of error codes reserved for formatting errors.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableErrorMinimum`
Marks beginning of the range of error codes reserved for errors related to executable files.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableNotLoadableError`
Executable is of a type that is not loadable in the current process.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableArchitectureMismatchError`
Executable does not provide an architecture compatible with the current process.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableRuntimeMismatchError`
Executable has Objective C runtime information incompatible with the current process.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableLoadError`
Executable cannot be loaded for some other reason, such as a problem with a library it depends on.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableLinkError`
Executable fails due to linking issues.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

`NSExecutableErrorMaximum`
Marks end of the range of error codes reserved for errors related to executable files.
Available in iPhone OS 2.0 and later.
Declared in `FoundationErrors.h`

Discussion
The constants in this enumeration are `NSError` code numbers in the Cocoa error domain (`NSCocoaErrorDomain`). Other frameworks, most notably the Application Kit, provide their own `NSCocoaErrorDomain` error codes.

The enumeration constants beginning with `NSFile` indicate file-system errors or errors related to file I/O operations. Use the key `NSFilePathErrorKey` or the `NSURLErrorKey` (whichever is appropriate) to access the file-system path or URL in the `userInfo` dictionary of the `NSError` object.

Declared In
`FoundationErrors.h`

NSNotFound

Defines a value that indicates that an item requested couldn't be found or doesn't exist.

```
enum {
   NSNotFound = 0x7fffffff
};
```

Constants

`NSNotFound`

A value that indicates that an item requested couldn't be found or doesn't exist.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

Discussion

`NSNotFound` is typically used by various methods and functions that search for items in serial data and return indices, such as characters in a string object or `id`s in an `NSArray` object.

Declared In

`NSObjCRuntime.h`

URL Loading System Error Codes

These values are returned as the error code property of an `NSError` object with the domain "NSURLErrorDomain".

```
typedef enum
{
    NSURLErrorUnknown = -1,
    NSURLErrorCancelled = -999,
    NSURLErrorBadURL = -1000,
    NSURLErrorTimedOut = -1001,
    NSURLErrorUnsupportedURL = -1002,
    NSURLErrorCannotFindHost = -1003,
    NSURLErrorCannotConnectToHost = -1004,
    NSURLErrorDataLengthExceedsMaximum = -1103,
    NSURLErrorNetworkConnectionLost = -1005,
    NSURLErrorDNSLookupFailed = -1006,
    NSURLErrorHTTPTooManyRedirects = -1007,
    NSURLErrorResourceUnavailable = -1008,
    NSURLErrorNotConnectedToInternet = -1009,
    NSURLErrorRedirectToNonExistentLocation = -1010,
    NSURLErrorBadServerResponse = -1011,
    NSURLErrorUserCancelledAuthentication = -1012,
    NSURLErrorUserAuthenticationRequired = -1013,
    NSURLErrorZeroByteResource = -1014,
    NSURLErrorFileDoesNotExist = -1100,
    NSURLErrorFileIsDirectory = -1101,
    NSURLErrorNoPermissionsToReadFile = -1102,
    NSURLErrorSecureConnectionFailed = -1200,
    NSURLErrorServerCertificateHasBadDate = -1201,
    NSURLErrorServerCertificateUntrusted = -1202,
    NSURLErrorServerCertificateHasUnknownRoot = -1203,
    NSURLErrorServerCertificateNotYetValid = -1204,
    NSURLErrorClientCertificateRejected = -1205,
    NSURLErrorCannotLoadFromNetwork = -2000,
    NSURLErrorCannotCreateFile = -3000,
    NSURLErrorCannotOpenFile = -3001,
    NSURLErrorCannotCloseFile = -3002,
    NSURLErrorCannotWriteToFile = -3003,
    NSURLErrorCannotRemoveFile = -3004,
    NSURLErrorCannotMoveFile = -3005,
    NSURLErrorDownloadDecodingFailedMidStream = -3006,
    NSURLErrorDownloadDecodingFailedToComplete = -3007
}
```

Constants

`NSURLErrorUnknown`

Returned when the URL Loading system encounters an error that it cannot interpret.

This can occur when an error originates from a lower level framework or library. Whenever this error code is received, it is a bug, and should be reported to Apple.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorCancelled`

Returned when an asynchronous load is canceled.

A Web Kit framework delegate will receive this error when it performs a cancel operation on a loading resource. Note that an `NSURLConnection` or `NSURLDownload` delegate will not receive this error if the download is canceled.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorBadURL`
Returned when a URL is sufficiently malformed that a URL request cannot be initiated

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorTimedOut`
Returned when an asynchronous operation times out.

`NSURLConnection` will send this error to its delegate when the timeoutInterval in NSURLRequest expires before a load can complete.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorUnsupportedURL`
Returned when a properly formed URL cannot be handled by the framework.

The most likely cause is that there is no available protocol handler for the URL.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorCannotFindHost`
Returned when the host name for a URL cannot be resolved.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorCannotConnectToHost`
Returned when an attempt to connect to a host has failed.

This can occur when a host name resolves, but the host is down or may not be accepting connections on a certain port.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorDataLengthExceedsMaximum`
Returned when the length of the resource data is too exceeds the maximum allowed.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorNetworkConnectionLost`
Returned when a client or server connection is severed in the middle of an in-progress load.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorDNSLookupFailed`
See `NSURLErrorCannotFindHost`

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorHTTPTooManyRedirects`
Returned when a redirect loop is detected or when the threshold for number of allowable redirects has been exceeded (currently 16).

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorResourceUnavailable`
Returned when a requested resource cannot be retrieved.

Examples are “file not found”, and data decoding problems that prevent data from being processed correctly.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorNotConnectedToInternet`
Returned when a network resource was requested, but an internet connection is not established and cannot be established automatically, either through a lack of connectivity, or by the user's choice not to make a network connection automatically.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorRedirectToNonExistentLocation`
Returned when a redirect is specified by way of server response code, but the server does not accompany this code with a redirect URL.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorBadServerResponse`
Returned when the URL Loading system receives bad data from the server.

This is equivalent to the “500 Server Error” message sent by HTTP servers.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorUserCancelledAuthentication`
Returned when an asynchronous request for authentication is cancelled by the user.

This is typically incurred by clicking a “Cancel” button in a username/password dialog, rather than the user making an attempt to authenticate.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorUserAuthenticationRequired`
Returned when authentication is required to access a resource.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorZeroByteResource`
Returned when a server reports that a URL has a non-zero content length, but terminates the network connection “gracefully” without sending any data.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorFileDoesNotExist`
Returned when a file does not exist.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorFileIsDirectory`
Returned when a request for an FTP file results in the server responding that the file is not a plain file, but a directory.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorNoPermissionsToReadFile`
Returned when a resource cannot be read due to insufficient permissions.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorSecureConnectionFailed`
Returned when an attempt to establish a secure connection fails for reasons which cannot be expressed more specifically.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorServerCertificateHasBadDate`
Returned when a server certificate has a date which indicates it has expired, or is not yet valid.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorServerCertificateUntrusted`
Returned when a server certificate is signed by a root server which is not trusted.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorServerCertificateHasUnknownRoot`
Returned when a server certificate is not signed by any root server.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorServerCertificateNotYetValid`
Returned when a server certificate is not yet valid.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorClientCertificateRejected`
Returned when a server certificate is rejected.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorCannotLoadFromNetwork`
Returned when a specific request to load an item only from the cache cannot be satisfied.

This error is sent at the point when the library would go to the network accept for the fact that is has been blocked from doing so by the “load only from cache” directive.

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

`NSURLErrorCannotCreateFile`
: Returned when `NSURLDownload` object was unable to create the downloaded file on disk due to a I/O failure.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorCannotOpenFile`
: Returned when `NSURLDownload` was unable to open the downloaded file on disk.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorCannotCloseFile`
: Returned when `NSURLDownload` was unable to close the downloaded file on disk.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorCannotWriteToFile`
: Returned when `NSURLDownload` was unable to write to the downloaded file on disk.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorCannotRemoveFile`
: Returned when `NSURLDownload` was unable to remove a downloaded file from disk.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorCannotMoveFile`
: Returned when `NSURLDownload` was unable to move a downloaded file on disk.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorDownloadDecodingFailedMidStream`
: Returned when `NSURLDownload` failed to decode an encoded file during the download.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

`NSURLErrorDownloadDecodingFailedToComplete`
: Returned when `NSURLDownload` failed to decode an encoded file after downloading.

  Available in iPhone OS 2.0 and later.

  Declared in `NSURLError.h`

Availability

Available in Mac OS X v10.2 with Safari 1.0 installed.

Available in Mac OS X v10.2.7 and later.

Declared In

`NSURLError.h`

Global Variables

Cocoa Error Domain

This constant defines the Cocoa error domain.

```
NSString *const NSCocoaErrorDomain;
```

Constants

`NSCocoaErrorDomain`

Application Kit and Foundation Kit errors.

Available in iPhone OS 2.0 and later.

Declared in `NSError.h`

Declared In

`FoundationErrors.h`

NSURL Domain

This error domain is defined for `NSURL`.

```
extern NSString * const NSURLErrorDomain;
```

Constants

`NSURLErrorDomain`

URL loading system errors

Available in iPhone OS 2.0 and later.

Declared in `NSURLError.h`

Declared In

`NSURLError.h`

Numeric Constants

NSDecimal Constants

Constants used by `NSDecimal`.

```
#define NSDecimalMaxSize (8)
#define NSDecimalNoScale SHRT_MAX
```

Constants

`NSDecimalMaxSize`

The maximum size of NSDecimal (page 1403).

Gives a precision of at least 38 decimal digits, 128 binary positions.

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

`NSDecimalNoScale`
Number of shorts in an NSDecimal (page 1403).

Available in iPhone OS 2.0 and later.

Declared in `NSDecimal.h`

Declared In
`NSDecimal.h`

NSInteger and NSUInteger Maximum and Minimum Values

Constants representing the maximum and minimum values of `NSInteger` and `NSUInteger`.

```
#define NSIntegerMax    LONG_MAX
#define NSIntegerMin    LONG_MIN
#define NSUIntegerMax   ULONG_MAX
```

Constants
`NSIntegerMax`
The maximum value for an `NSInteger`.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSIntegerMin`
The minimum value for an `NSInteger`.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSUIntegerMax`
The maximum value for an `NSUInteger`.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

Declared In
`NSObjCRuntime.h`

Exceptions

General Exception Names

Exceptions defined by `NSException`.

```
extern NSString *NSGenericException;
extern NSString *NSRangeException;
extern NSString *NSInvalidArgumentException;
extern NSString *NSInternalInconsistencyException;
extern NSString *NSMallocException;
extern NSString *NSObjectInaccessibleException;
extern NSString *NSObjectNotAvailableException;
extern NSString *NSDestinationInvalidException;
extern NSString *NSPortTimeoutException;
extern NSString *NSInvalidSendPortException;
extern NSString *NSInvalidReceivePortException;
extern NSString *NSPortSendException;
extern NSString *NSPortReceiveException;
extern NSString *NSOldStyleException;
```

Constants

`NSGenericException`

A generic name for an exception.

You should typically use a more specific exception name.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSRangeException`

Name of an exception that occurs when attempting to access outside the bounds of some data, such as beyond the end of a string.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSInvalidArgumentException`

Name of an exception that occurs when you pass an invalid argument to a method, such as a `nil` pointer where a non-`nil` object is required.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSInternalInconsistencyException`

Name of an exception that occurs when an internal assertion fails and implies an unexpected condition within the called code.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSMallocException`

Obsolete; not currently used.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSObjectInaccessibleException`

Name of an exception that occurs when a remote object is accessed from a thread that should not access it.

See `NSConnection`'s `enableMultipleThreads`.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSObjectNotAvailableException`
Name of an exception that occurs when the remote side of the `NSConnection` refused to send the message to the object because the object has never been vended.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSDestinationInvalidException`
Name of an exception that occurs when an internal assertion fails and implies an unexpected condition within the distributed objects.

This is a distributed objects–specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSPortTimeoutException`
Name of an exception that occurs when a timeout set on a port expires during a send or receive operation.

This is a distributed objects–specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSInvalidSendPortException`
Name of an exception that occurs when the send port of an `NSConnection` has become invalid.

This is a distributed objects–specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSInvalidReceivePortException`
Name of an exception that occurs when the receive port of an `NSConnection` has become invalid.

This is a distributed objects–specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSPortSendException`
Generic error occurred on send.

This is an `NSPort`-specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSPortReceiveException`
Generic error occurred on receive.

This is an `NSPort`-specific exception.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

`NSOldStyleException`
No longer used.

Available in iPhone OS 2.0 and later.

Declared in `NSException.h`

Declared In
NSException.h

Version Numbers

Foundation Version Number

Version of the Foundation framework in the current environment.

```
double NSFoundationVersionNumber;
```

Constants
NSFoundationVersionNumber
The version of the Foundation framework in the current environment.
Available in iPhone OS 2.0 and later.
Declared in NSObjCRuntime.h

Declared In
NSObjCRuntime.h

Foundation Framework Version Numbers

Constants to define Foundation Framework version numbers.

```
#define NSFoundationVersionNumber10_0    397.40
#define NSFoundationVersionNumber10_1    425.00
#define NSFoundationVersionNumber10_1_1    425.00
#define NSFoundationVersionNumber10_1_2    425.00
#define NSFoundationVersionNumber10_1_3    425.00
#define NSFoundationVersionNumber10_1_4    425.00
#define NSFoundationVersionNumber10_2    462.00
#define NSFoundationVersionNumber10_2_1    462.00
#define NSFoundationVersionNumber10_2_2    462.00
#define NSFoundationVersionNumber10_2_3    462.00
#define NSFoundationVersionNumber10_2_4    462.00
#define NSFoundationVersionNumber10_2_5    462.00
#define NSFoundationVersionNumber10_2_6    462.00
#define NSFoundationVersionNumber10_2_7    462.70
#define NSFoundationVersionNumber10_2_8    462.70
#define NSFoundationVersionNumber10_3    500.00
#define NSFoundationVersionNumber10_3_1    500.00
#define NSFoundationVersionNumber10_3_2    500.30
#define NSFoundationVersionNumber10_3_3    500.54
#define NSFoundationVersionNumber10_3_4    500.56
#define NSFoundationVersionNumber10_3_5    500.56
#define NSFoundationVersionNumber10_3_6    500.56
#define NSFoundationVersionNumber10_3_7    500.56
#define NSFoundationVersionNumber10_3_8    500.56
#define NSFoundationVersionNumber10_3_9    500.58
#define NSFoundationVersionNumber10_4    567.00
#define NSFoundationVersionNumber10_4_1    567.00
#define NSFoundationVersionNumber10_4_2    567.12
#define NSFoundationVersionNumber10_4_3    567.21
#define NSFoundationVersionNumber10_4_4_Intel    567.23
#define NSFoundationVersionNumber10_4_4_PowerPC    567.21
#define NSFoundationVersionNumber10_4_5    567.25
#define NSFoundationVersionNumber10_4_6    567.26
#define NSFoundationVersionNumber10_4_7    567.27
```

Constants

`NSFoundationVersionNumber10_0`

Foundation version released in Mac OS X version 10.0.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_1`

Foundation version released in Mac OS X version 10.1.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_1_1`

Foundation version released in Mac OS X version 10.1.1.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_1_2`

Foundation version released in Mac OS X version 10.1.2.

Available in iPhone OS 2.0 and later.

Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_1_3`
Foundation version released in Mac OS X version 10.1.3.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_1_4`
Foundation version released in Mac OS X version 10.1.4.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2`
Foundation version released in Mac OS X version 10.2.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_1`
Foundation version released in Mac OS X version 10.2.1.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_2`
Foundation version released in Mac OS X version 10.2.2.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_3`
Foundation version released in Mac OS X version 10.2.3.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_4`
Foundation version released in Mac OS X version 10.2.4.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_5`
Foundation version released in Mac OS X version 10.2.5.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_6`
Foundation version released in Mac OS X version 10.2.6.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_7`
Foundation version released in Mac OS X version 10.2.7.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_2_8`
Foundation version released in Mac OS X version 10.2.8.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3`
Foundation version released in Mac OS X version 10.3.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_1`
Foundation version released in Mac OS X version 10.3.1.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_2`
Foundation version released in Mac OS X version 10.3.2.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_3`
Foundation version released in Mac OS X version 10.3.3.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_4`
Foundation version released in Mac OS X version 10.3.4.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_5`
Foundation version released in Mac OS X version 10.3.5.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_6`
Foundation version released in Mac OS X version 10.3.6.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_7`
Foundation version released in Mac OS X version 10.3.7.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_8`
Foundation version released in Mac OS X version 10.3.8.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_3_9`
Foundation version released in Mac OS X version 10.3.9.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4`
Foundation version released in Mac OS X version 10.4.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_1`
Foundation version released in Mac OS X version 10.4.1.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_2`
Foundation version released in Mac OS X version 10.4.2.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_3`
Foundation version released in Mac OS X version 10.4.3.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_4_Intel`
Foundation version released in Mac OS X version 10.4.4 for Intel.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_4_PowerPC`
Foundation version released in Mac OS X version 10.4.4 for PowerPC.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_5`
Foundation version released in Mac OS X version 10.4.5.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_6`
Foundation version released in Mac OS X version 10.4.6.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

`NSFoundationVersionNumber10_4_7`
Foundation version released in Mac OS X version 10.4.7.
Available in iPhone OS 2.0 and later.
Declared in `NSObjCRuntime.h`

Declared In
`NSObjCRuntime.h`

Document Revision History

This table describes the changes to *Foundation Framework Reference*.

Date	Notes
2008-06-27	Updated for iPhone OS.
2007-10-31	Updated for Mac OS X v10.5. Updated framework illustrations.
2007-04-16	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a collection of separate documents.

Document Revision History

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z