Foundation Framework Reference

2010-05-20

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 NSAttributedString.h NSAutoreleasePool.h NSBundle.h NSByteOrder.h NSCache.h NSCalendar.h NSCharacterSet.h NSCoder.h NSComparisonPredicate.h NSCompoundPredicate.h NSData.h NSDate.h NSDateFormatter.h NSDecimal.h NSDecimalNumber.h NSDictionary.h NSEnumerator.h NSError.h NSException.h NSExpression.h NSFileHandle.h NSFileManager.h NSFileWrapper.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
NSOrthography.h
NSPathUtilities.h
NSPort.h
NSPredicate.h
NSProcessInfo.h
NSPropertyList.h
NSProxy.h
NSRange.h
NSRegularExpression.h
NSRunLoop.h
NSScanner.h
NSSet.h
NSSortDescriptor.h
NSStream.h
NSString.h
NSTextCheckingResult.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
NSUndoManager.h
NSUserDefaults.h
NSValue.h
NSValueTransformer.h
NSXMLParser.h
NSZone.h

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 30) for a list of those classes that make up the Foundation framework.

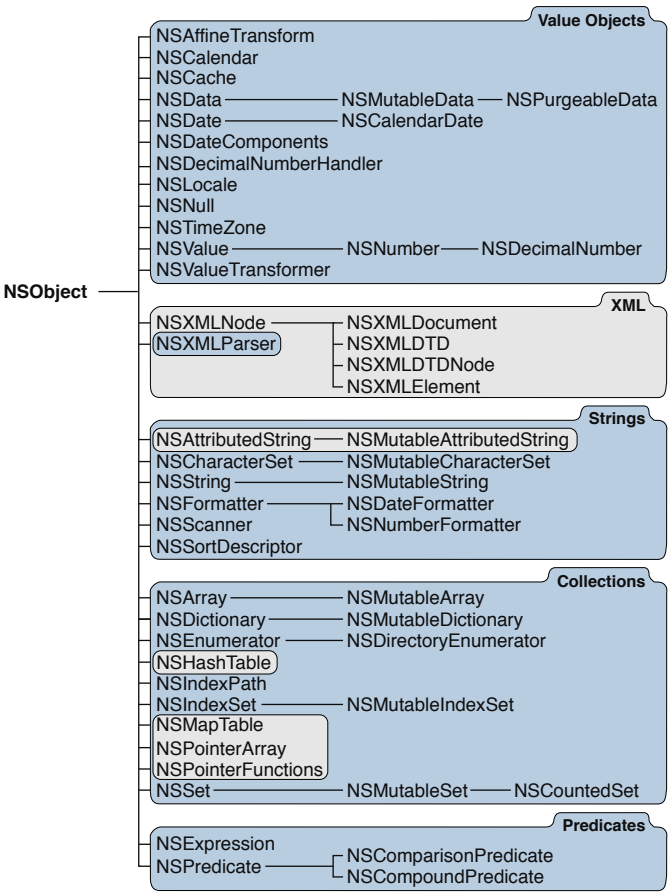
The Foundation framework introduces several paradigms to avoid confusion in common situations, and to introduce a level of consistency across class hierarchies. This consistency is done with some standard policies, such as that for object ownership (that is, who is responsible for disposing of objects), 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

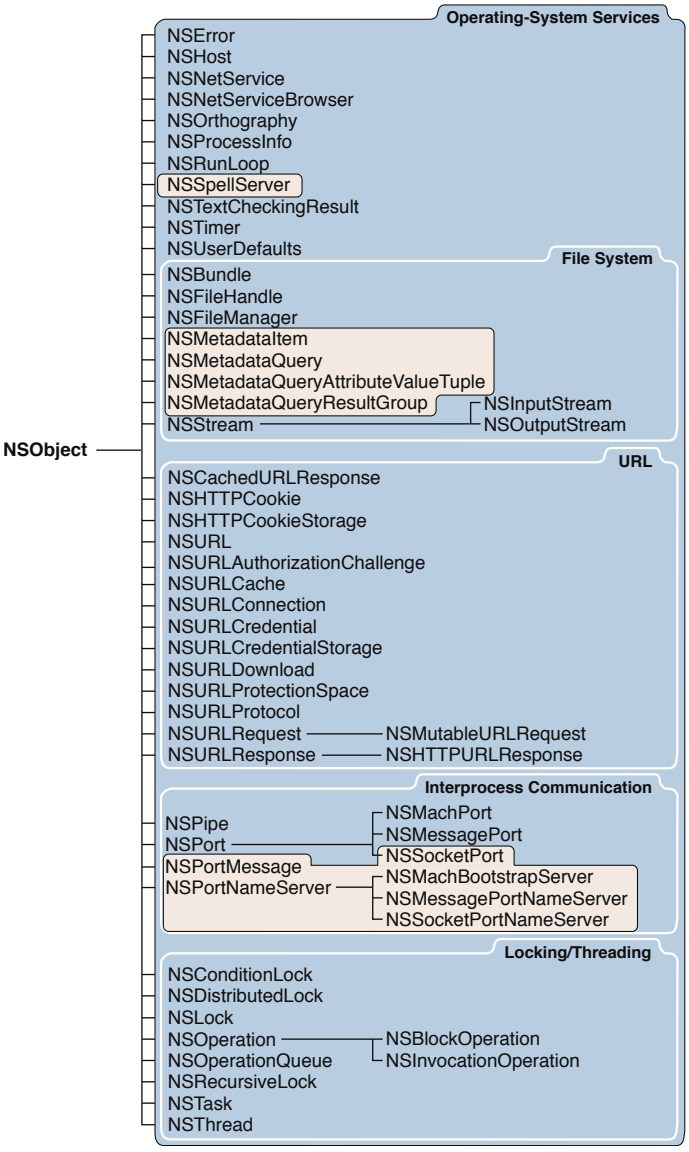
The Foundation class hierarchy is rooted in the Foundation framework's `NSObject` class (see Figure I-1 (page 30)). The remainder of the Foundation framework consists of several related groups of classes as well as a few individual classes. Many of the groups form what are called class clusters—abstract 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 iOS; gray-shaded areas include classes that are available in Mac OS X only.

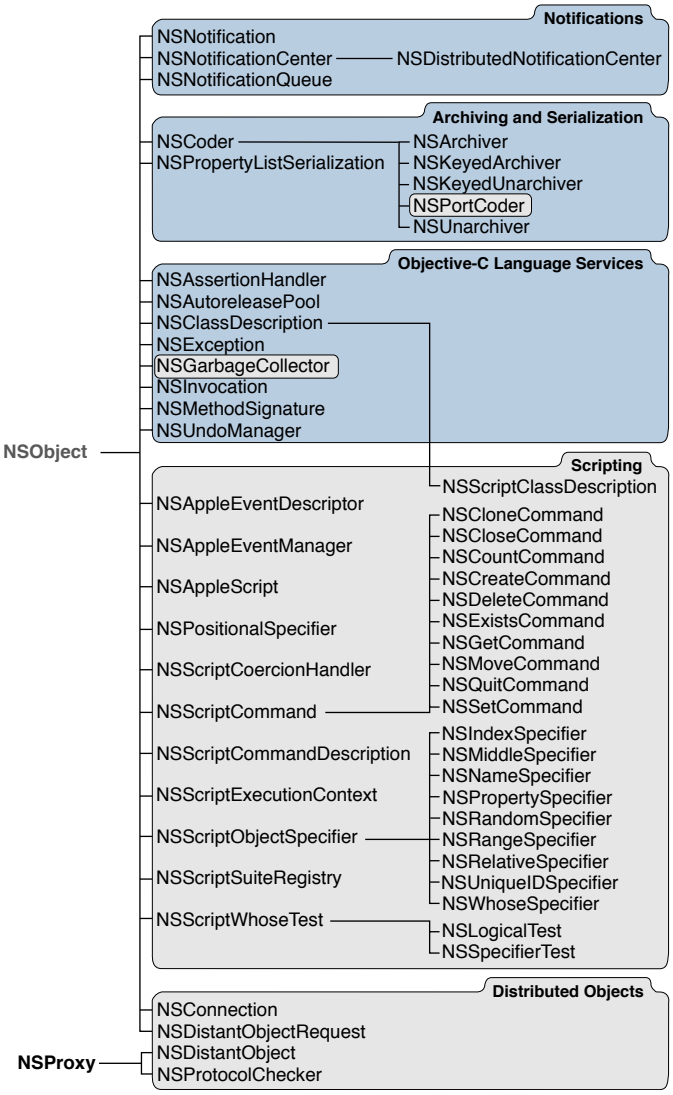
Figure I-1 Cocoa Objective-C Hierarchy for Foundation



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 iOS 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 Key-Value Coding Programming Guide Property List Programming Guide Predicate Programming Guide
Related sample code	GKRocket MoviePlayer ScrollViewSuite SpeakHere ToolbarSearch

Overview

`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 52) and objectAtIndex: (page 72)—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 73) and reverseObjectEnumerator (page 76) 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 61) 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 1632) message, as declared in the NSObject protocol. Another method, indexOfObjectIdenticalTo: (page 65), 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: (page 56) method allows you to create a new array from an existing array filtered using a predicate (see *Predicate Programming Guide*).

NSArray's makeObjectsPerformSelector: (page 72) and makeObjectsPerformSelector:withObject: (page 72) 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 80) and sortedArrayUsingFunction:context: (page 78), extract a subset of the array (subarrayWithRange: (page 81)), or concatenate the elements of an array of NSString objects into a single string (componentsJoinedByString: (page 51)). In addition, you can compare two arrays using the isEqualToArray: (page 71) and firstObjectCommonWithArray: (page 57) methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with arrayByAddingObject: (page 50) and arrayByAddingObjectsFromArray: (page 50).

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

There is typically little 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.

- 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 52) and `objectAtIndex:` (page 72). 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 971) 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 1638) to each object that you add to your collection and `release` (page 1636) 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
`copyWithZone:` (page 1554)

NSMutableCopying
`mutableCopyWithZone:` (page 1614)

Tasks

Creating an Array

`+ array` (page 44)
Creates and returns an empty array.

`+ arrayWithArray:` (page 45)
Creates and returns an array containing the objects in another given array.

`+ arrayWithContentsOfFile:` (page 45)
Creates and returns an array containing the contents of the file specified by a given path.

`+ arrayWithContentsOfURL:` (page 46)
Creates and returns an array containing the contents specified by a given URL.

`+ arrayWithObject:` (page 46)
Creates and returns an array containing a given object.

`+ arrayWithObjects:` (page 47)
Creates and returns an array containing the objects in the argument list.

`+ arrayWithObjects:count:` (page 48)
Creates and returns an array that includes a given number of objects from a given C array.

Initializing an Array

`- initWithArray:` (page 67)
Initializes a newly allocated array by placing in it the objects contained in a given array.

`- initWithArray:copyItems:` (page 68)
Initializes a newly allocated array using *anArray* as the source of data objects for the array.

`- initWithContentsOfFile:` (page 68)
Initializes a newly allocated array with the contents of the file specified by a given path.

`- initWithContentsOfURL:` (page 69)
Initializes a newly allocated array with the contents of the location specified by a given URL.

`- initWithObjects:` (page 70)
Initializes a newly allocated array by placing in it the objects in the argument list.

`- initWithObjects:count:` (page 70)
Initializes a newly allocated array to include a given number of objects from a given C array.

Querying an Array

- `containsObject:` (page 51)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `count` (page 52)
  Returns the number of objects currently in the receiver.
- `getObjects:range:` (page 58)
  Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.
- `lastObject` (page 71)
  Returns the object in the array with the highest index value.
- `objectAtIndex:` (page 72)
  Returns the object located at *index*.
- `objectsAtIndexes:` (page 74)
  Returns an array containing the objects in the receiver at the indexes specified by a given index set.
- `objectEnumerator` (page 73)
  Returns an enumerator object that lets you access each object in the receiver.
- `reverseObjectEnumerator` (page 76)
  Returns an enumerator object that lets you access each object in the receiver, in reverse order.
- `getObjects:` (page 57) Deprecated in iOS 4.0
  Copies all the objects contained in the receiver to *aBuffer*.

Finding Objects in an Array

- `indexOfObject:` (page 61)
  Returns the lowest index whose corresponding array value is equal to a given object.
- `indexOfObject:inRange:` (page 62)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectIdenticalTo:` (page 65)
  Returns the lowest index whose corresponding array value is identical to a given object.
- `indexOfObjectIdenticalTo:inRange:` (page 65)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectPassingTest:` (page 66)
  Returns the index of the first object in the receiver that passes a test in a given Block.
- `indexOfObjectWithOptions:passingTest:` (page 66)
  Returns the index of an object in the receiver that passes a test in a given Block for a given set of enumeration options.
- `indexOfObjectAtIndexes:options:passingTest:` (page 64)
  Returns the index, from a given set of indexes, of the first object in the receiver that passes a test in a given Block for a given set of enumeration options.
- `indexesOfObjectsPassingTest:` (page 60)
  Returns the indexes of objects in the receiver that pass a test in a given Block.

- `indexesOfObjectsWithOptions:passingTest:` (page 60)
  Returns the indexes of objects in the receiver that pass a test in a given Block for a given set of enumeration options.
- `indexesOfObjectsAtIndexes:options:passingTest:` (page 59)
  Returns the indexes, from a given set of indexes, of objects in the receiver that pass a test in a given Block for a given set of enumeration options.
- `indexOfObject:inSortedRange:options:usingComparator:` (page 62)
  Returns the index, within a specified range, of an object compared with elements in the receiver using a given `NSComparator` block.

Sending Messages to Elements

- `makeObjectsPerformSelector:` (page 72)
  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 72)
  Sends the *aSelector* message to each object in the array, starting with the first object and continuing through the array to the last object.
- `enumerateObjectsUsingBlock:` (page 55)
  Executes a given block using each object in the receiver, starting with the first object and continuing through the array to the last object.
- `enumerateObjectsWithOptions:usingBlock:` (page 56)
  Executes a given block using each object in the receiver.
- `enumerateObjectsAtIndexes:options:usingBlock:` (page 54)
  Executes a given block using the objects in the receiver at the specified indexes.

Comparing Arrays

- `firstObjectCommonWithArray:` (page 57)
  Returns the first object contained in the receiver that's equal to an object in another given array.
- `isEqualToArray:` (page 71)
  Compares the receiving array to another array.

Deriving New Arrays

- `arrayByAddingObject:` (page 50)
  Returns a new array that is a copy of the receiver with a given object added to the end.
- `arrayByAddingObjectsFromArray:` (page 50)
  Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.
- `filteredArrayUsingPredicate:` (page 56)
  Evaluates a given predicate against each object in the receiver and returns a new array containing the objects for which the predicate returns true.

- `subarrayWithRange:` (page 81)
  Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

Sorting

- `sortedArrayHint` (page 77)
  Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 79).
- `sortedArrayUsingFunction:context:` (page 78)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingFunction:context:hint:` (page 79)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingDescriptors:` (page 77)
  Returns a copy of the receiver sorted as specified by a given array of sort descriptors.
- `sortedArrayUsingSelector:` (page 80)
  Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given selector.
- `sortedArrayUsingComparator:` (page 77)
  Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given `NSComparator` Block.
- `sortedArrayWithOptions:usingComparator:` (page 80)
  Returns an array that lists the receiver's elements in ascending order, as determined by the comparison method specified by a given `NSComparator` Block.

Working with String Elements

- `componentsJoinedByString:` (page 51)
  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 52)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:` (page 53)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 53)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `writeToFile:atomically:` (page 82)
  Writes the contents of the receiver to a file at a given path.

- `writeToURL:atomically:` (page 83)
  Writes the contents of the receiver to the location specified by a given URL.

Collecting Paths

- `pathsMatchingExtensions:` (page 74)
  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 48)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 75)
  Raises an exception.
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 49)
  Registers *anObserver* to receive key value observer notifications for the specified *keypath* relative to the objects at *indexes*.
- `removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 75)
  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 76)
  Invokes `setValue:forKey:` on each of the receiver's items using the specified *value* and *key*.
- `valueForKey:` (page 82)
  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 iOS 2.0 and later.

See Also
`+ arrayWithObject:` (page 46)
`+ arrayWithObjects:` (page 47)

Related Sample Code
BonjourWeb
ToolbarSearch

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 iOS 2.0 and later.

See Also
`+ arrayWithObjects:` (page 47)
`- initWithObjects:` (page 70)

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 82) 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 iOS 2.0 and later.

See Also
`- writeToFile:atomically:` (page 82)

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 83) 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 iOS 2.0 and later.

See Also
`- writeToURL:atomically:` (page 83)

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 iOS 2.0 and later.

See Also
`+ array` (page 44)
`+ arrayWithObjects:` (page 47)

Related Sample Code
BonjourWeb
GKTank
ToolbarSearch
WiTap

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 iOS 2.0 and later.

See Also
`+ array` (page 44)
`+ arrayWithObject:` (page 46)

Related Sample Code
ToolbarSearch

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*. For example:

```
NSString *strings[3];
strings[0] = @"First";
strings[1] = @"Second";
strings[2] = @"Third";

NSArray *stringsArray = [NSArray arrayWithObjects:strings count:2];
// strings array contains { @"First", @"Second" }
```

Availability

Available in iOS 2.0 and later.

See Also

- `getObjects:` (page 57)
- `getObjects:range:` (page 58)

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 1600).

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

options
A combination of `NSKeyValueObservingOptions` (page 1605) values that specifies what is included in observation notifications.

context
Arbitrary data that is passed to *observer* in `observeValueForKeyPath:ofObject:change:context:` (page 1600).

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 iOS 2.0 and later.

See Also
- `removeObserver:forKeyPath:` (page 75)
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 49)

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
```

Parameters

anObserver
The observer.

indexes
The index set.

keyPath
The key path, relative to the receiver, to be observed.

options
The options to be included in the notification.

context
The context passed to the notifications.

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 1598).

Availability
Available in iOS 2.0 and later.

See Also
`- removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 75)

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 iOS 2.0 and later.

See Also
`addObject:` (page 731) (`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 iOS 2.0 and later.

See Also
`addObjectsFromArray:` (page 731) (`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 iOS 2.0 and later.

See Also
componentsSeparatedByString: (page 1213) (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 1632) message to each of the receiver's objects (and passing *anObject* as the parameter to each isEqual: message).

Availability
Available in iOS 2.0 and later.

See Also
- `indexOfObject:` (page 61)
- `indexOfObjectIdenticalTo:` (page 65)

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 iOS 2.0 and later.

See Also
- `objectAtIndex:` (page 72)

Related Sample Code
CryptoExercise
SpeakHere

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 iOS 2.0 and later.

See Also
- `descriptionWithLocale:` (page 53)
- `descriptionWithLocale:indent:` (page 53)

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 53).

Availability
Available in iOS 2.0 and later.

See Also
- `description` (page 52)
- `descriptionWithLocale:indent:` (page 53)

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 53), 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 52) method.

Availability
Available in iOS 2.0 and later.

See Also
- `- description` (page 52)
- `- descriptionWithLocale:` (page 53)

Declared In
`NSArray.h`

enumerateObjectsAtIndexes:options:usingBlock:

Executes a given block using the objects in the receiver at the specified indexes.

```
- (void)enumerateObjectsAtIndexes:(NSIndexSet *)indexSet
    options:(NSEnumerationOptions)opts
    usingBlock:(void (^)(id obj, NSUInteger idx, BOOL *stop))block
```

Parameters

indexSet
The indexes of the objects over which to enumerate.

opts
A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

block
The block to apply to elements in the array.

The block takes three arguments:

obj
The element in the array.

idx
The index of the element in the array.

stop
A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

Discussion
By default, the enumeration starts with the first object and continues serially through the array to the last element specified by *indexSet*. You can specify `NSEnumerationConcurrent` and/or `NSEnumerationReverse` as enumeration options to modify this behavior.

Important: If the Block parameter or the *indexSet* is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateObjectsUsingBlock:` (page 55)
- `makeObjectsPerformSelector:` (page 72)
- `makeObjectsPerformSelector:withObject:` (page 72)

Declared In
`NSArray.h`

enumerateObjectsUsingBlock:

Executes a given block using each object in the receiver, starting with the first object and continuing through the array to the last object.

```
- (void)enumerateObjectsUsingBlock:(void (^)(id obj, NSUInteger idx, BOOL
    *stop))block
```

Parameters

block

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

Discussion
If the Block parameter is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateObjectsWithOptions:usingBlock:` (page 56)
- `makeObjectsPerformSelector:` (page 72)
- `makeObjectsPerformSelector:withObject:` (page 72)

Declared In
`NSArray.h`

enumerateObjectsWithOptions:usingBlock:

Executes a given block using each object in the receiver.

```
- (void)enumerateObjectsWithOptions:(NSEnumerationOptions)opts
    usingBlock:(void (^)(id obj, NSUInteger idx, BOOL *stop))block
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

block

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

Discussion

By default, the enumeration starts with the first object and continues serially through the array to the last object. You can specify `NSEnumerationConcurrent` and/or `NSEnumerationReverse` as enumeration options to modify this behavior.

Important: If the Block parameter is `nil` this method will raise an exception.

Availability

Available in iOS 4.0 and later.

See Also

- `enumerateObjectsUsingBlock:` (page 55)
- `makeObjectsPerformSelector:` (page 72)
- `makeObjectsPerformSelector:withObject:` (page 72)

Declared In

`NSArray.h`

filteredArrayUsingPredicate:

Evaluates a given predicate against each object in the receiver and returns a new array containing the objects for which the predicate returns true.

```
- (NSArray *)filteredArrayUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate

The predicate against which to evaluate the receiver's elements.

Return Value

A new array containing the objects in the receiver for which *predicate* returns true.

Discussion

For more details, see *Predicate Programming Guide*.

Availability

Available in iOS 3.0 and later.

Related Sample Code

ToolbarSearch

Declared In

NSPredicate.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 1632) to check for object equality.

Availability

Available in iOS 2.0 and later.

See Also

- containsObject: (page 51)

Declared In

NSArray.h

getObjects:

Copies all the objects contained in the receiver to *aBuffer*. (Deprecated in iOS 4.0.)

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

Parameters

aBuffer

A C array of objects of size at least the count of the receiver.

Discussion
The method copies into *aBuffer* all the objects in the receiver; the size of the buffer must therefore be at least the count of the receiver multiplied by the size of an object reference, as shown in the following example (note that this is just an example, you should typically not create a buffer simply to iterate over the contents of an array):

```
NSArray *mArray = // ...;
id *objects;

NSUInteger count = [mArray count];
objects = malloc(sizeof(id) * count);

[mArray getObjects:objects];

for (i = 0; i < count; i++) {
    NSLog(@"object at index %d: %@", i, objects[i]);
}
free(objects);
```

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 4.0.

See Also
`+ arrayWithObjects:count:` (page 48)

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
```

Parameters

aBuffer
A C array of objects of size at least the length of the range specified by *aRange*.

aRange
A range within the bounds of the receiver.

If the location plus the length of the range is greater than the count of the receiver, this method raises an `NSRangeException` (page 1773).

Discussion
The method copies into *aBuffer* the objects in the receiver in the range specified by *aRange*; the size of the buffer must therefore be at least the length of the range multiplied by the size of an object reference, as shown in the following example (this is solely for illustration—you should typically not create a buffer simply to iterate over the contents of an array):

```
NSArray *mArray = // an array with at least six elements...;
id *objects;

NSRange range = NSMakeRange(2, 4);
objects = malloc(sizeof(id) * range.length);
```

```
[mArray getObjects:objects range:range];

for (i = 0; i < range.length; i++) {
    NSLog(@"objects: %@", objects[i]);
}
free(objects);
```

Availability
Available in iOS 2.0 and later.

See Also
`+ arrayWithObjects:count:` (page 48)

Declared In
`NSArray.h`

indexesOfObjectsAtIndexes:options:passingTest:

Returns the indexes, from a given set of indexes, of objects in the receiver that pass a test in a given Block for a given set of enumeration options.

```
- (NSIndexSet *)indexesOfObjectsAtIndexes:(NSIndexSet *)indexSet
    options:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger
    idx, BOOL *stop))predicate
```

Parameters

indexSet

The indexes of the objects over which to enumerate.

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The indexes from *indexSet* whose corresponding values in the receiver pass the test specified by *predicate*.

Discussion
By default, the enumeration starts with the first object and continues serially through the array to the last element specified by *indexSet*. You can specify `NSEnumerationConcurrent` and/or `NSEnumerationReverse` as enumeration options to modify this behavior.

Important: If the Block parameter or the *indexSet* is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

Declared In
`NSArray.h`

indexesOfObjectsPassingTest:

Returns the indexes of objects in the receiver that pass a test in a given Block.

```
- (NSIndexSet *)indexesOfObjectsPassingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL
    *stop))predicate
```

Parameters

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The indexes whose corresponding values in the receiver pass the test specified by *predicate*.

Availability
Available in iOS 4.0 and later.

Declared In
`NSArray.h`

indexesOfObjectsWithOptions:passingTest:

Returns the indexes of objects in the receiver that pass a test in a given Block for a given set of enumeration options.

```
- (NSIndexSet *)indexesOfObjectsWithOptions:(NSEnumerationOptions)opts
    passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The indexes whose corresponding values in the receiver pass the test specified by *predicate*.

Discussion
By default, the enumeration starts with the first object and continues serially through the array to the last object. You can specify `NSEnumerationConcurrent` (page 1758) and/or `NSEnumerationReverse` (page 1758) as enumeration options to modify this behavior.

Important: If the Block parameter is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

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 1632) returns `YES`.

Important: If *anObject* is `nil` an exception is raised.

Availability
Available in iOS 2.0 and later.

See Also
- `containsObject:` (page 51)
- `indexOfObjectIdenticalTo:` (page 65)

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 1632) returns `YES`.

Availability
Available in iOS 2.0 and later.

See Also
- `containsObject:` (page 51)
- `indexOfObjectIdenticalTo:inRange:` (page 65)

Declared In
`NSArray.h`

indexOfObject:inSortedRange:options:usingComparator:

Returns the index, within a specified range, of an object compared with elements in the receiver using a given `NSComparator` block.

```
- (NSUInteger)indexOfObject:(id)obj
    inSortedRange:(NSRange)r
    options:(NSBinarySearchingOptions)opts
    usingComparator:(NSComparator)cmp
```

Parameters

obj

An object for which to search in the receiver.

If this value is `nil`, throws an `NSInvalidArgumentException` (page 1773).

r

The range within the receiver to search for *obj*.

If *r* exceeds the bounds of the receiver (if the location plus length of the range is greater than the count of the receiver), throws an `NSRangeException` (page 1773).

opts

Options for the search. For possible values, see "NSBinarySearchingOptions" (page 83).

If you specify both `NSBinarySearchingFirstEqual` (page 84) and `NSBinarySearchingLastEqual` (page 84), throws an `NSInvalidArgumentException`.

cmp

A comparator block used to compare the object *obj* with elements in the receiver.

If this value is `NULL`, throws an `NSInvalidArgumentException` (page 1773).

Return Value

If the `NSBinarySearchingInsertionIndex` (page 84) option is not specified:

- If the *obj* is found and neither `NSBinarySearchingFirstEqual` (page 84) nor `NSBinarySearchingLastEqual` (page 84) is specified, returns an arbitrary matching object's index.
- If the `NSBinarySearchingFirstEqual` (page 84) option is also specified, returns the lowest index of equal objects.
- If the `NSBinarySearchingLastEqual` (page 84) option is also specified, returns the highest index of equal objects.
- If the object is not found, returns `NSNotFound`.

If the `NSBinarySearchingInsertionIndex` (page 84) option is specified, returns the index at which you should insert *obj* in order to maintain a sorted array:

- If the *obj* is found and neither `NSBinarySearchingFirstEqual` (page 84) nor `NSBinarySearchingLastEqual` (page 84) is specified, returns any equal or one larger index than any matching object's index.
- If the `NSBinarySearchingFirstEqual` (page 84) option is also specified, returns the lowest index of equal objects.
- If the `NSBinarySearchingLastEqual` (page 84) option is also specified, returns the highest index of equal objects.
- If the object is not found, returns the index of the least greater object, or the index at the end of the array if the object is larger than all other elements.

Special Considerations

The elements in the receiver must have already been sorted using the comparator *cmp*. If the array is not sorted, the result is undefined.

Availability
Available in iOS 4.0 and later.

Declared In
NSArray.h

indexOfObjectAtIndexes:options:passingTest:

Returns the index, from a given set of indexes, of the first object in the receiver that passes a test in a given Block for a given set of enumeration options.

```
- (NSUInteger)indexOfObjectAtIndexes:(NSIndexSet *)indexSet
    options:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger
    idx, BOOL *stop))predicate
```

Parameters

indexSet

The indexes of the objects over which to enumerate.

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to YES to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The lowest index whose corresponding value in the receiver passes the test specified by *predicate*. If no objects in the receiver pass the test, returns NSNotFound.

Discussion
By default, the enumeration starts with the first object and continues serially through the array to the last element specified by *indexSet*. You can specify NSEnumerationConcurrent (page 1758) and/or NSEnumerationReverse (page 1758) as enumeration options to modify this behavior.

Important: If the Block parameter or *indexSet* is nil this method will raise an exception.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
- containsObject: (page 51)
- indexOfObject: (page 61)

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 iOS 2.0 and later.

See Also
- `containsObject:` (page 51)
- `indexOfObject:inRange:` (page 62)

Declared In
`NSArray.h`

indexOfObjectPassingTest:

Returns the index of the first object in the receiver that passes a test in a given Block.

```
- (NSUInteger)indexOfObjectPassingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL
    *stop))predicate
```

Parameters

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The lowest index whose corresponding value in the receiver passes the test specified by *predicate*. If no objects in the receiver pass the test, returns `NSNotFound`.

Discussion
If the Block parameter is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

Declared In
`NSArray.h`

indexOfObjectWithOptions:passingTest:

Returns the index of an object in the receiver that passes a test in a given Block for a given set of enumeration options.

```
- (NSUInteger)indexOfObjectWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL
    (^)(id obj, NSUInteger idx, BOOL *stop))predicate
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

predicate

The block to apply to elements in the array.

The block takes three arguments:

obj

The element in the array.

idx

The index of the element in the array.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The index whose corresponding value in the receiver passes the test specified by *predicate* and `opts`. If the *opts* bitmask specifies reverse order, then the last item that matches is returned. Otherwise, the index of the first matching object is returned. If no objects in the receiver pass the test, returns `NSNotFound`.

Discussion
By default, the enumeration starts with the first object and continues serially through the array to the last object. You can specify `NSEnumerationConcurrent` and/or `NSEnumerationReverse` as enumeration options to modify this behavior.

Important: If the Block parameter is `nil` this method will raise an exception.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
`+ arrayWithObject:` (page 46)
`- initWithObjects:` (page 70)

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:` (page 954) 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 iOS 2.0 and later.

See Also
`- initWithArray:` (page 67)
`+ arrayWithObject:` (page 46)
`- initWithObjects:` (page 70)

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 82) 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 iOS 2.0 and later.

See Also

`+ arrayWithContentsOfFile:` (page 45)

`- writeToFile:atomically:` (page 82)

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 83) 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 iOS 2.0 and later.

See Also

`+ arrayWithContentsOfURL:` (page 46)

`- writeToURL:atomically:` (page 83)

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 iOS 2.0 and later.

See Also

- `initWithObjects:count:` (page 70)
+ `arrayWithObjects:` (page 47)
- `initWithArray:` (page 67)

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 iOS 2.0 and later.

See Also
- `initWithObjects:` (page 70)
+ `arrayWithObjects:` (page 47)
- `initWithArray:` (page 67)

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 1632) test.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`removeLastObject` (page 736) (`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 iOS 2.0 and later.

See Also

`- makeObjectsPerformSelector:withObject:` (page 72)

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 iOS 2.0 and later.

See Also

`- makeObjectsPerformSelector:` (page 72)

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` (page 1773) is raised.

Availability

Available in iOS 2.0 and later.

See Also

- `count` (page 52)
- `objectsAtIndexes:` (page 74)

Related Sample Code

MoviePlayer

SpeakHere

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 iOS 2.0 and later.

See Also
- `reverseObjectEnumerator` (page 76)
`nextObject` (page 424) (`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` (page 1773) exception if any location in *indexes* exceeds the bounds of the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `count` (page 52)
- `objectAtIndex:` (page 72)

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 iOS 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 iOS 2.0 and later.

See Also
- addObserver:forKeyPath:options:context: (page 48)
- removeObserver:fromObjectsAtIndexes:forKeyPath: (page 75)

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
```

Parameters

anObserver
The observer.

indexes
The index set.

keyPath
The key path, relative to the receiver, to be observed.

Discussion
This is not merely a convenience method; invoking this method is potentially much faster than repeatedly invoking `removeObserver:forKeyPath:` (page 1601).

Availability
Available in iOS 2.0 and later.

See Also
`- addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 49)

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 iOS 2.0 and later.

See Also
`- objectEnumerator` (page 73)
`nextObject` (page 424) (`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
```

Parameters

`value`
The object value.

`key`
The key to store the value.

Availability
Available in iOS 2.0 and later.

See Also
- `valueForKey:` (page 82)

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 79).

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

Availability
Available in iOS 2.0 and later.

See Also
- `sortedArrayUsingFunction:context:hint:` (page 79)

Declared In
`NSArray.h`

sortedArrayUsingComparator:

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

```
- (NSArray *)sortedArrayUsingComparator:(NSComparator)cmptr
```

Parameters

cmptr

A comparator block.

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

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also

- `sortedArrayUsingSelector:` (page 80)
- `sortedArrayUsingFunction:context:` (page 78)
- `sortedArrayUsingFunction:context:hint:` (page 79)

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 iOS 2.0 and later.

See Also
- `sortedArrayUsingDescriptors:` (page 77)
- `sortedArrayUsingFunction:context:hint:` (page 79)
- `sortedArrayUsingSelector:` (page 80)

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 78), 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 77). 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 77) 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 79) 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 iOS 2.0 and later.

See Also
- `sortedArrayUsingDescriptors:` (page 77)

- sortedArrayUsingFunction:context: (page 78)
- sortedArrayUsingSelector: (page 80)

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 1207) 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 iOS 2.0 and later.

See Also
- sortedArrayUsingDescriptors: (page 77)
- sortedArrayUsingFunction:context: (page 78)
- sortedArrayUsingFunction:context:hint: (page 79)

Declared In
NSArray.h

sortedArrayWithOptions:usingComparator:

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

```
- (NSArray *)sortedArrayWithOptions:(NSSortOptions)opts
    usingComparator:(NSComparator)cmptr
```

Parameters

opts

A bitmask that specifies the options for the sort (whether it should be performed concurrently and whether it should be performed stably).

cmptr

A comparator block.

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

Availability
Available in iOS 4.0 and later.

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 iOS 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
```

Parameters

key

The key to retrieve.

Return Value

The value of the retrieved key.

Discussion

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

Availability

Available in iOS 2.0 and later.

See Also

`- setValue:forKey:` (page 76)

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 1267) 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 45) or the instance method `initWithContentsOfFile:` (page 68). 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 iOS 2.0 and later.

See Also
- initWithContentsOfFile: (page 68)

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 46) or the instance method initWithContentsOfURL: (page 69).

Availability
Available in iOS 2.0 and later.

See Also
- initWithContentsOfURL: (page 69)

Declared In
NSArray.h

Constants

NSBinarySearchingOptions

Options for searches and insertions using indexOfObject:inSortedRange:options:usingComparator: (page 62).

```
enum {
    NSBinarySearchingFirstEqual = (1 << 8),
    NSBinarySearchingLastEqual = (1 << 9),
    NSBinarySearchingInsertionIndex = (1 << 10),
};
typedef NSUInteger NSBinarySearchingOptions;
```

Constants

`NSBinarySearchingFirstEqual`

Specifies that the search should return the first object in the range that is equal to the given object.

Available in iOS 4.0 and later.

Declared in `NSArray.h`.

`NSBinarySearchingLastEqual`

Specifies that the search should return the last object in the range that is equal to the given object.

Available in iOS 4.0 and later.

Declared in `NSArray.h`.

`NSBinarySearchingInsertionIndex`

Returns the index at which you should insert the object in order to maintain a sorted array.

Available in iOS 4.0 and later.

Declared in `NSArray.h`.

NSAssertionHandler Class Reference

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

Overview

`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 87) and `handleFailureInFunction:file:lineNumber:description:` (page 86) messages respectively to the current assertion handler. The assertion handler for the current thread is obtained using the `currentHandler` (page 86) class method. See `NSAssertionHandlerKey` (page 88) if you need to customize the behavior of `NSAssertionHandler`.

Tasks

Handling Assertion Failures

`+ currentHandler` (page 86)
Returns the `NSAssertionHandler` object associated with the current thread.

`- handleFailureInFunction:file:lineNumber:description:` (page 86)
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 87)
  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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`NSException.h`

Constants

The string constants for exceptions are listed and described in the `Exceptions` section of the *Foundation Constants Reference*.

NSAssertionHandlerKey

This constant refers to a key in the thread dictionary of the per-thread assertion handler object

```
NSString * const NSAssertionHandlerKey;
```

Constants

`NSAssertionHandlerKey`

A key with a corresponding value in the thread dictionary.

If you need to customize the behavior of `NSAssertionHandler`, create a subclass, overriding `handleFailureInMethod:object:file:lineNumber:description:` (page 87) and `handleFailureInFunction:file:lineNumber:description:` (page 86), and install your instance into the current thread's attributes dictionary with this key.

NSAttributedString Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.2 and later.
Declared in	Foundation/NSAttributedString.h
Companion guide	Attributed String Programming Guide

Overview

`NSAttributedString` objects manage character strings and associated sets of attributes (for example, font and kerning) that apply to individual characters or ranges of characters in the string. An association of characters and their attributes is called an attributed string. The cluster's two public classes, `NSAttributedString` and `NSMutableAttributedString`, declare the programmatic interface for read-only attributed strings and modifiable attributed strings, respectively. The Foundation framework defines only the basic functionality for attributed strings; in Mac OS X, additional methods supporting RTF, graphics attributes, and drawing attributed strings are described in NSAttributedString Additions, found in the Application Kit. The Application Kit also uses a subclass of `NSMutableAttributedString`, called NSTextStorage, to provide the storage for the Application Kit's extended text-handling system.

In Mac OS X, the Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes.

An attributed string identifies attributes by name, storing a value under the name in an `NSDictionary` object. In Mac OS X, standard attribute keys are described in the "Constants" section of *NSAttributedString Application Kit Additions Reference*. You can also assign any attribute name/value pair you wish to a range of characters—it is up to your application to interpret custom attributes (see *Attributed String Programming Guide*). If you are using attributed strings with the Core Text framework, you can also use the attribute keys defined by that framework.

Note that the default font for `NSAttributedString` objects is Helvetica 12-point, which differs from the Mac OS X system font Lucida Grande, so you may wish to create the string with non-default attributes suitable for your application using, for example, `initWithString:attributes:` (page 98).

Be aware that `isEqual:` comparison among `NSAttributedString` objects compares for exact equality, including not only literal character-by-character string equality but also equality of all attributes, which is not likely to be achieved in the case of many attributes such as attachments, lists, and tables, for example.

iOS Note: In iOS, this class is used primarily in conjunction with the Core Text framework.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating an NSAttributedString Object

- `initWithString:` (page 98)
  Returns an `NSAttributedString` object initialized with the characters of a given string and no attribute information.
- `initWithAttributedString:` (page 97)
  Returns an `NSAttributedString` object initialized with the characters and attributes of another given attributed string.
- `initWithString:attributes:` (page 98)
  Returns an `NSAttributedString` object initialized with a given string and attributes.

Retrieving Character Information

- `string` (page 99)
  Returns the character contents of the receiver as an `NSString` object.
- `length` (page 99)
  Returns the length of the receiver's string object.

Retrieving Attribute Information

- `attributesAtIndex:effectiveRange:` (page 94)
  Returns the attributes for the character at a given index.
- `attributesAtIndex:longestEffectiveRange:inRange:` (page 94)
  Returns the attributes for the character at a given index, and by reference the range over which the attributes apply.
- `attribute:atIndex:effectiveRange:` (page 91)
  Returns the value for an attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.
- `attribute:atIndex:longestEffectiveRange:inRange:` (page 92)
  Returns the value for the attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

Comparing Attributed Strings

- `isEqualToAttributedString:` (page 98)
  Returns a Boolean value that indicates whether the receiver is equal to another given attributed string.

Extracting a Substring

- `attributedSubstringFromRange:` (page 93)
  Returns an `NSAttributedString` object consisting of the characters and attributes within a given range in the receiver.

Enumerating over Attributes in a String

- `enumerateAttribute:inRange:options:usingBlock:` (page 95)
  Executes the Block for the specified attribute run in the specified range.
- `enumerateAttributesInRange:options:usingBlock:` (page 96)
  Executes the Block for each attribute in the range.

Instance Methods

attribute:atIndex:effectiveRange:

Returns the value for an attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

```
- (id)attribute:(NSString *)attributeName atIndex:(NSUInteger)index
    effectiveRange:(NSRangePointer)aRange
```

Parameters

attributeName

The name of an attribute.

index

The index for which to return attributes. This value must not exceed the bounds of the receiver.

aRange

If non-NULL:

- If the named attribute exists at *index*, upon return *aRange* contains a range over which the named attribute's value applies.
- If the named attribute does not exist at *index*, upon return *aRange* contains the range over which the attribute does not exist.

The range isn't necessarily the maximum range covered by *attributeName*, and its extent is implementation-dependent. If you need the maximum range, use `attribute:atIndex:longestEffectiveRange:inRange:` (page 92). If you don't need this value, pass NULL.

Return Value

The value for the attribute named *attributeName* of the character at index *index*, or nil if there is no such attribute.

Discussion

Raises an NSRangeException if *index* lies beyond the end of the receiver's characters.

For information about where to find the attribute keys for the returned dictionary, see the overview section of this document.

Availability

Available in iOS 3.2 and later.

See Also

`- attributesAtIndex:effectiveRange:` (page 94)

Declared In

NSAttributedString.h

attribute:atIndex:longestEffectiveRange:inRange:

Returns the value for the attribute with a given name of the character at a given index, and by reference the range over which the attribute applies.

```
- (id)attribute:(NSString *)attributeName atIndex:(NSUInteger)index
    longestEffectiveRange:(NSRangePointer)aRange inRange:(NSRange)rangeLimit
```

Parameters

attributeName

The name of an attribute.

index

The index at which to test for *attributeName*.

aRange

If non-NULL:

- If the named attribute exists at *index*, upon return *aRange* contains the full range over which the value of the named attribute is the same as that at *index*, clipped to *rangeLimit*.
- If the named attribute does not exist at *index*, upon return *aRange* contains the full range over which the attribute does not exist, clipped to *rangeLimit*.

If you don't need this value, pass NULL.

rangeLimit

The range over which to search for continuous presence of *attributeName*. This value must not exceed the bounds of the receiver.

Return Value
The value for the attribute named *attributeName* of the character at *index*, or nil if there is no such attribute.

Discussion
Raises an NSRangeException if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

If you don't need the longest effective range, it's far more efficient to use the attribute:atIndex:effectiveRange: (page 91) method to retrieve the attribute value.

For information about where to find the attribute keys for the returned dictionary, see the overview section of this document.

Availability
Available in iOS 3.2 and later.

See Also
- attributesAtIndex:longestEffectiveRange:inRange: (page 94)

Declared In
NSAttributedString.h

attributedSubstringFromRange:

Returns an NSAttributedString object consisting of the characters and attributes within a given range in the receiver.

```
- (NSAttributedString *)attributedSubstringFromRange:(NSRange)aRange
```

Parameters

aRange

The range from which to create a new attributed string. *aRange* must lie within the bounds of the receiver.

Return Value
An NSAttributedString object consisting of the characters and attributes within *aRange* in the receiver.

Discussion
Raises an NSRangeException if any part of *aRange* lies beyond the end of the receiver's characters. This method treats the length of the string as a valid range value that returns an empty string.

Availability
Available in iOS 3.2 and later.

Declared In
NSAttributedString.h

attributesAtIndex:effectiveRange:

Returns the attributes for the character at a given index.

```
- (NSDictionary *)attributesAtIndex:(NSUInteger)index
    effectiveRange:(NSRangePointer)aRange
```

Parameters

index

The index for which to return attributes. This value must lie within the bounds of the receiver.

aRange

Upon return, the range over which the attributes and values are the same as those at *index*. This range isn't necessarily the maximum range covered, and its extent is implementation-dependent. If you need the maximum range, use `attributesAtIndex:longestEffectiveRange:inRange:` (page 94). If you don't need this value, pass `NULL`.

Return Value
The attributes for the character at *index*.

Discussion
Raises an `NSRangeException` if *index* lies beyond the end of the receiver's characters.

For information about where to find the attribute keys for the returned dictionary, see the overview section of this document.

Availability
Available in iOS 3.2 and later.

See Also
`- attribute:atIndex:effectiveRange:` (page 91)

Declared In
NSAttributedString.h

attributesAtIndex:longestEffectiveRange:inRange:

Returns the attributes for the character at a given index, and by reference the range over which the attributes apply.

```
- (NSDictionary *)attributesAtIndex:(NSUInteger)index
    longestEffectiveRange:(NSRangePointer)aRange inRange:(NSRange)rangeLimit
```

Parameters

index

The index for which to return attributes. This value must not exceed the bounds of the receiver.

aRange

If non-NULL, upon return contains the maximum range over which the attributes and values are the same as those at *index*, clipped to *rangeLimit*.

rangeLimit

The range over which to search for continuous presence of the attributes at *index*. This value must not exceed the bounds of the receiver.

Discussion

Raises an NSRangeException if *index* or any part of *rangeLimit* lies beyond the end of the receiver's characters.

If you don't need the range information, it's far more efficient to use the attributesAtIndex:effectiveRange: (page 94) method to retrieve the attribute value.

For information about where to find the attribute keys for the returned dictionary, see the overview section of this document.

Availability

Available in iOS 3.2 and later.

See Also

- attribute:atIndex:longestEffectiveRange:inRange: (page 92)

Declared In

NSAttributedString.h

enumerateAttribute:inRange:options:usingBlock:

Executes the Block for the specified attribute run in the specified range.

```
- (void)enumerateAttribute:(NSString *)attrName inRange:(NSRange)enumerationRange
    options:(NSAttributedStringEnumerationOptions)opts usingBlock:(void (^)(id
   value, NSRange range, BOOL *stop))block
```

Parameters

attrName

The name of an attribute.

enumerationRange

If non-NULL, contains the maximum range over which the attributes and values are enumerated, clipped to *enumerationRange*.

opts

The options used by the enumeration. The values can be combined using C-bitwise OR. The values are described in "NSAttributedStringEnumerationOptions" (page 100).

block

The Block to apply to ranges of the attribute in the attributed string.

The Block takes three arguments:

value

The *attrName* value.

range

An NSRange indicating the run of the attribute.

stop

A reference to a Boolean value. The block can set the value to YES to stop further processing of the set. The stop argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

Discussion

If this method is sent to an instance of NSMutableAttributedString, mutation (deletion, addition, or change) is allowed, as long as it is within the range provided to the block; after a mutation, the enumeration continues with the range immediately following the processed range, after the length of the processed range is adjusted for the mutation. (The enumerator basically assumes any change in length occurs in the specified range.)

For example, if *block* is called with a range starting at location N, and the block deletes all the characters in the supplied range, the next call will also pass *N* as the index of the range.

Availability

Available in iOS 4.0 and later.

Declared In

NSAttributedString.h

enumerateAttributesInRange:options:usingBlock:

Executes the Block for each attribute in the range.

```
- (void)enumerateAttributesInRange:(NSRange)enumerationRange
    options:(NSAttributedStringEnumerationOptions)opts usingBlock:(void
    (^)(NSDictionary *attrs, NSRange range, BOOL *stop))block
```

Parameters

enumerationRange

If non-NULL, contains the maximum range over which the attributes and values are enumerated, clipped to *enumerationRange*.

opts

The options used by the enumeration. The values can be combined using C-bitwise OR. The values are described in "NSAttributedStringEnumerationOptions" (page 100).

block

The Block to apply to ranges of the attribute in the attributed string.

The Block takes three arguments:

attrs

An NSDictionary that contains the attributes for the range.

range

An NSRange indicating the run of the attribute.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

Discussion
If this method is sent to an instance of `NSMutableAttributedString`, mutation (deletion, addition, or change) is allowed, as long as it is within the range provided to the block; after a mutation, the enumeration continues with the range immediately following the processed range, after the length of the processed range is adjusted for the mutation. (The enumerator basically assumes any change in length occurs in the specified range.)

For example, if *block* is called with a range starting at location `N`, and the block deletes all the characters in the supplied range, the next call will also pass *N* as the index of the range.

Availability
Available in iOS 4.0 and later.

Declared In
`NSAttributedString.h`

initWithAttributedString:

Returns an `NSAttributedString` object initialized with the characters and attributes of another given attributed string.

```
- (id)initWithAttributedString:(NSAttributedString *)attributedString
```

Parameters

attributedString

An attributed string.

Return Value
An `NSAttributedString` object initialized with the characters and attributes of *attributedString*. The returned object might be different than the original receiver.

Availability
Available in iOS 3.2 and later.

Declared In
`NSAttributedString.h`

initWithString:

Returns an `NSAttributedString` object initialized with the characters of a given string and no attribute information.

```
- (id)initWithString:(NSString *)aString
```

Parameters

aString

The characters for the new object.

Return Value

An `NSAttributedString` object initialized with the characters of *aString* and no attribute information The returned object might be different than the original receiver.

Availability

Available in iOS 3.2 and later.

Declared In

`NSAttributedString.h`

initWithString:attributes:

Returns an `NSAttributedString` object initialized with a given string and attributes.

```
- (id)initWithString:(NSString *)aString attributes:(NSDictionary *)attributes
```

Parameters

aString

The string for the new attributed string.

attributes

The attributes for the new attributed string. For information about where to find the attribute keys you can include in this dictionary, see the overview section of this document.

Discussion

Returns an `NSAttributedString` object initialized with the characters of *aString* and the attributes of *attributes*. The returned object might be different from the original receiver.

Availability

Available in iOS 3.2 and later.

Declared In

`NSAttributedString.h`

isEqualToAttributedString:

Returns a Boolean value that indicates whether the receiver is equal to another given attributed string.

```
- (BOOL)isEqualToAttributedString:(NSAttributedString *)otherString
```

Parameters

otherString

The attributed string with which to compare the receiver.

Return Value
`YES` if the receiver is equal to *`otherString`*, otherwise `NO`.

Discussion
Attributed strings must match in both characters and attributes to be equal.

Availability
Available in iOS 3.2 and later.

Declared In
`NSAttributedString.h`

length

Returns the length of the receiver's string object.

```
- (NSUInteger)length
```

Availability
Available in iOS 3.2 and later.

See Also
`length` (page 1246) (NSString)

Declared In
`NSAttributedString.h`

string

Returns the character contents of the receiver as an `NSString` object.

```
- (NSString *)string
```

Return Value
The character contents of the receiver as an `NSString` object.

Discussion
This method doesn't strip out attachment characters; use `NSText`'s `string` method to extract just the linguistically significant characters.

For performance reasons, this method returns the current backing store of the attributed string object. If you want to maintain a snapshot of this as you manipulate the returned string, you should make a copy of the appropriate substring.

This primitive method must guarantee efficient access to an attributed string's characters; subclasses should implement it to execute in O(1) time.

Availability
Available in iOS 3.2 and later.

Declared In
`NSAttributedString.h`

Constants

Standard attribute keys are described in the "Constants" section of *NSAttributedString Application Kit Additions Reference*.

NSAttributedStringEnumerationOptions

These constants describe the options available to the `enumerateAttribute:inRange:options:usingBlock:` (page 95) and `enumerateAttributesInRange:options:usingBlock:` (page 96) methods.

```
enum {
  NSAttributedStringEnumerationReverse = (1UL << 1),
  NSAttributedStringEnumerationLongestEffectiveRangeNotRequired = (1UL << 20)
};
typedef NSUInteger NSAttributedStringEnumerationOptions;
```

Constants

`NSAttributedStringEnumerationReverse`

Causes the enumeration to occur in reverse.

Available in iOS 4.0 and later.

Declared in `NSAttributedString.h`.

`NSAttributedStringEnumerationLongestEffectiveRangeNotRequired`

If NSAttributedStringEnumerationLongestEffectiveRangeNotRequired option is supplied, then the longest effective range computation is not performed; the blocks may be invoked with consecutive attribute runs that have the same value.

Available in iOS 4.0 and later.

Declared in `NSAttributedString.h`.

NSAutoreleasePool Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSAutoreleasePool.h
Companion guide	Memory Management Programming Guide
Related sample code	aurioTouch CryptoExercise FastEnumerationSample ScrollViewSuite SpeakHere

Overview

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

In a reference-counted environment (as opposed to one which uses garbage collection), an `NSAutoreleasePool` object contains objects that have received an `autorelease` (page 1629) message and when drained it sends a `release` (page 1636) message to each of those objects. Thus, sending `autorelease` (page 1629) instead of `release` (page 1636) to an object extends the lifetime of that object at least until the pool itself is drained (it may be longer if the object is subsequently retained). An object can be put into the same pool several times, in which case it receives a `release` (page 1636) message for each time it was put into the pool.

In a reference counted environment, 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. In this situation, your program will typically log suitable warning messages.

The Application Kit creates an autorelease pool on the main thread at the beginning of every cycle of the event loop, and drains it at the end, thereby releasing any autoreleased objects generated while processing an event. If you use the Application Kit, you therefore typically don't have to create your own pools. If your application creates a lot of temporary autoreleased objects within the event loop, however, 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 `drain` (page 104) (or `release` (page 105)—to understand the difference, see "Garbage Collection" (page 102)). Since you cannot retain an autorelease pool (or autorelease it—see `retain` (page 105) and `autorelease` (page 104)), draining a pool ultimately has the effect of deallocating it. 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.

Each thread (including the main thread) maintains its own stack of `NSAutoreleasePool` objects (see "Threads" (page 102)). 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 drains 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 drain 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 1314).

Garbage Collection

In a garbage-collected environment, there is no need for autorelease pools. You may, however, write a framework that is designed to work in both a garbage-collected and reference-counted environment. In this case, you can use autorelease pools to hint to the collector that collection may be appropriate. In a garbage-collected environment, sending a `drain` (page 104) message to a pool triggers garbage collection if necessary; `release` (page 105), however, is a no-op. In a reference-counted environment, `drain` (page 104) has the same effect as `release` (page 105). Typically, therefore, you should use `drain` (page 104) instead of `release` (page 105).

Tasks

Managing a Pool

- `release` (page 105)
  Releases and pops the receiver.

- `drain` (page 104)
  In a reference-counted environment, releases and pops the receiver; in a garbage-collected environment, triggers garbage collection if the memory allocated since the last collection is greater than the current threshold.
- `autorelease` (page 104)
  Raises an exception.
- `retain` (page 105)
  Raises an exception.

Adding an Object to a Pool

+ `addObject:` (page 103)
  Adds a given object to the active autorelease pool in the current thread.
- `addObject:` (page 104)
  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 1636) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 1629) to *object* instead.

Availability

Available in iOS 2.0 and later.

See Also

- `addObject:` (page 104)

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; when the pool is deallocated, the object will receive a `release` (page 1636) message for each time it was added.

Normally you don't invoke this method directly—you send `autorelease` (page 1629) to *object* instead.

Availability

Available in iOS 2.0 and later.

See Also

`+ addObject:` (page 103)

Declared In

`NSAutoreleasePool.h`

autorelease

Raises an exception.

```
- (id)autorelease
```

Return Value

`self`.

Discussion

In a reference-counted environment, this method raises an exception.

drain

In a reference-counted environment, releases and pops the receiver; in a garbage-collected environment, triggers garbage collection if the memory allocated since the last collection is greater than the current threshold.

```
- (void)drain
```

Discussion
In a reference-counted environment, this method behaves the same as `release` (page 1636). Since an autorelease pool cannot be retained (see `retain` (page 105)), this therefore causes the receiver to be deallocated. When an autorelease pool is deallocated, it sends a `release` (page 1636) 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 1636) message for each time it was added.

In a garbage-collected environment, this method ultimately calls `objc_collect_if_needed`.

Special Considerations

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
FastEnumerationSample

Declared In
`NSAutoreleasePool.h`

release

Releases and pops the receiver.

```
- (void)release
```

Discussion
In a reference-counted environment, since an autorelease pool cannot be retained (see `retain` (page 105)), this method causes the receiver to be deallocated. When an autorelease pool is deallocated, it sends a `release` (page 1636) 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 1636) message for each time it was added.

In a garbage-collected environment, this method is a no-op.

Special Considerations

You should typically use `drain` (page 104) instead of `release`.

See Also
- `drain` (page 104)

retain

Raises an exception.

```
- (id)retain
```

Return Value
`self`.

Discussion
In a reference-counted environment, this method raises an exception.

NSBlockOperation Class Reference

Inherits from	NSOperation : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Threading Programming Guide

Overview

The `NSBlockOperation` class is a concrete subclass of `NSOperation` that manages the concurrent execution of one or more blocks. You can use this object to execute several blocks at once without having to create separate operation objects for each. When executing more than one block, the operation itself is considered finished only when all blocks have finished executing.

Blocks added to a block operation are dispatched with default priority to an appropriate work queue. The blocks themselves should not make any assumptions about the configuration of their execution environment.

For more information about blocks, see *Blocks Programming Topics*.

Tasks

Managing the Blocks in the Operation

`+ blockOperationWithBlock:` (page 108)
Creates and returns an `NSBlockOperation` object and adds the specified block to it.

`- addExecutionBlock:` (page 108)
Adds the specified block to the receiver's list of blocks to perform.

`- executionBlocks` (page 108)
Returns an array containing the blocks associated with the receiver.

Class Methods

blockOperationWithBlock:

Creates and returns an `NSBlockOperation` object and adds the specified block to it.

```
+ (id)blockOperationWithBlock:(void (^)(void))block
```

Parameters

block

The block to add to the new block operation object's list. The block should take no parameters and have no return value.

Return Value
A new block operation object.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

Instance Methods

addExecutionBlock:

Adds the specified block to the receiver's list of blocks to perform.

```
- (void)addExecutionBlock:(void (^)(void))block
```

Parameters

block

The block to add to the receiver's list. The block should take no parameters and have no return value.

Discussion
The specified block should not make any assumptions about is execution environment.

Calling this method while the receiver is executing or has already finished causes an `NSInvalidArgumentException` exception to be thrown.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

executionBlocks

Returns an array containing the blocks associated with the receiver.

```
- (NSArray *)executionBlocks
```

Return Value
An array of blocks. The blocks in this array are copies of the ones originally added using the `addExecutionBlock:` method.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

NSBundle Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSBundle.h
Companion guides	Bundle Programming Guide Resource Programming Guide
Related sample code	aurioTouch GKRocket GKTank MoviePlayer ScrollViewSuite

Overview

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, plug-ins.

Although bundle structures vary depending on the target platform and the type of bundle you are building, the `NSBundle` class hides this underlying structure in most (but not all) cases. Many of the methods you use to load resources from a bundle automatically locate the appropriate starting directory and look for resources in known places. For information about application bundle structures (for Mac OS X and iOS), see *Bundle Programming Guide*. For information about the structure of framework bundles, see *Framework Programming Guide*. For information about the structure of Mac OS X plug-ins, see *Code Loading Programming Topics*.

For additional information about how to load nib files and images in a Mac OS X application, see *NSBundle Additions Reference*. For information about how to load nib files in an iOS application, see *NSBundle UIKit Additions Reference*.

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

`+ bundleWithURL:` (page 118)
Returns an `NSBundle` object that corresponds to the specified file URL.

`+ bundleWithPath:` (page 117)
Returns an `NSBundle` object that corresponds to the specified directory.

`- initWithURL:` (page 127)
Returns an `NSBundle` object initialized to correspond to the specified file URL.

`- initWithPath:` (page 127)
Returns an `NSBundle` object initialized to correspond to the specified directory.

Getting an NSBundle

`+ bundleForClass:` (page 116)
Returns the `NSBundle` object with which the specified class is associated.

`+ bundleWithIdentifier:` (page 117)
Returns the previously created `NSBundle` instance that has the specified bundle identifier.

`+ mainBundle` (page 118)
Returns the `NSBundle` object that corresponds to the directory where the current application executable is located.

`+ allBundles` (page 115)
Returns an array of all the application's non-framework bundles.

`+ allFrameworks` (page 116)
Returns an array of all of the application's bundles that represent frameworks.

Getting a Bundled Class

`- classNamed:` (page 124)
Returns the `Class` object for the specified name.

`- principalClass` (page 138)
Returns the receiver's principal class.

Finding Resources

`- URLForResource:withExtension:subdirectory:` (page 143)
Returns the file URL for the resource file identified by the specified name and extension and residing in a given bundle directory.

`+ pathForResource:ofType:inDirectory:` (page 119)
Returns the full pathname for the resource file identified by the specified name and extension and residing in a given bundle directory.

- `URLForResource:withExtension:` (page 143)
  Returns the file URL for the resource identified by the specified name and file extension.
- `pathForResource:ofType:` (page 133)
  Returns the full pathname for the resource identified by the specified name and file extension.
- `URLsForResourcesWithExtension:subdirectory:` (page 145)
  Returns the file URL for the resource identified by the specified name and file extension and located in the specified bundle subdirectory.
- `pathForResource:ofType:inDirectory:` (page 134)
  Returns the full pathname for the resource identified by the specified name and file extension and located in the specified bundle subdirectory.
- `URLForResource:withExtension:subdirectory:localization:` (page 144)
  Returns the file URL for the resource identified by the specified name and file extension, located in the specified bundle subdirectory, and limited to global resources and those associated with the specified localization.
- `pathForResource:ofType:inDirectory:forLocalization:` (page 134)
  Returns the full pathname for the resource identified by the specified 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 120)
  Returns an array containing the pathnames for all bundle resources having the specified extension and residing in the bundle directory at the specified path.
- `pathsForResourcesOfType:inDirectory:` (page 135)
  Returns an array containing the pathnames for all bundle resources having the specified filename extension and residing in the resource subdirectory.
- `URLsForResourcesWithExtension:subdirectory:localization:` (page 146)
  Returns an array containing the file URLs 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.
- `pathsForResourcesOfType:inDirectory:forLocalization:` (page 136)
  Returns an array containing the file 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.
- `URLForResource:withExtension:subdirectory:inBundleWithURL:` (page 122)
  Creates and returns a file URL for the resource with the specified name and extension in the specified bundle.
- `URLsForResourcesWithExtension:subdirectory:inBundleWithURL:` (page 122)
  Returns an array containing the file URLs for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, within the specified bundle.
- `resourcePath` (page 140)
  Returns the full pathname of the receiving bundle's subdirectory containing resources.

Getting the Bundle Directory

- `bundleURL` (page 124)
  Returns the full URL of the receiver's bundle directory.

- `bundlePath` (page 124)
  Returns the full pathname of the receiver's bundle directory.

Getting Bundle Information

- `bundleIdentifier` (page 123)
  Returns the receiver's bundle identifier.
- `infoDictionary` (page 126)
  Returns a dictionary that contains information about the receiver.
- `objectForInfoDictionaryKey:` (page 132)
  Returns the value associated with the specified key in the receiver's information property list.
- `builtInPlugInsURL` (page 123)
  Returns the file URL of the receiver's subdirectory containing plug-ins.
- `builtInPlugInsPath` (page 123)
  Returns the full pathname of the receiver's subdirectory containing plug-ins.
- `executableURL` (page 126)
  Returns the file URL of the receiver's executable file.
- `executablePath` (page 126)
  Returns the full pathname of the receiver's executable file.
- `URLForAuxiliaryExecutable:` (page 142)
  Returns the file URL of the executable with the specified name in the receiver's bundle.
- `pathForAuxiliaryExecutable:` (page 132)
  Returns the full pathname of the executable with the specified name in the receiver's bundle.
- `privateFrameworksURL` (page 139)
  Returns the file URL of the receiver's subdirectory containing private frameworks.
- `privateFrameworksPath` (page 139)
  Returns the full pathname of the receiver's subdirectory containing private frameworks.
- `sharedFrameworksURL` (page 141)
  Returns the file URL of the receiver's subdirectory containing shared frameworks.
- `sharedFrameworksPath` (page 140)
  Returns the full pathname of the receiver's subdirectory containing shared frameworks.
- `sharedSupportURL` (page 141)
  Returns the file URL of the receiver's subdirectory containing shared support files.
- `sharedSupportPath` (page 141)
  Returns the full pathname of the receiver's subdirectory containing shared support files.
- `resourceURL` (page 140)
  Returns the file URL of the receiver's subdirectory containing resource files.

Managing Localized Resources

- `localizedStringForKey:value:table:` (page 131)
  Returns a localized version of the string designated by the specified key and residing in the specified table.

Loading a Bundle's Code

- `executableArchitectures` (page 125)
  Returns an array of numbers indicating the architecture types supported by the bundle's executable.
- `preflightAndReturnError:` (page 137)
  Returns a Boolean value indicating whether the bundle's executable code could be loaded successfully.
- `load` (page 128)
  Dynamically loads the bundle's executable code into a running program, if the code has not already been loaded.
- `loadAndReturnError:` (page 129)
  Loads the bundle's executable code and returns any errors.
- `isLoaded` (page 128)
  Obtains information about the load status of a bundle.
- `unload` (page 142)
  Unloads the code associated with the receiver.

Managing Localizations

+ `preferredLocalizationsFromArray:` (page 121)
  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 121)
  Returns the localizations that a bundle object would prefer, given the specified bundle and user preference localizations.
- `localizations` (page 130)
  Returns a list of all the localizations contained within the receiver's bundle.
- `developmentLocalization` (page 125)
  Returns the localization used to create the bundle.
- `preferredLocalizations` (page 137)
  Returns an array of strings indicating the actual localizations contained in the receiver's bundle.
- `localizedInfoDictionary` (page 130)
  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 iOS 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 iOS 2.0 and later.

Declared In
`NSBundle.h`

bundleForClass:

Returns the `NSBundle` object with which the specified 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 iOS 2.0 and later.

See Also
`+ mainBundle` (page 118)
`+ bundleWithPath:` (page 117)

Declared In
`NSBundle.h`

bundleWithIdentifier:

Returns the previously created `NSBundle` instance that has the specified 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 116) method.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`+ mainBundle` (page 118)
`+ bundleForClass:` (page 116)
`- initWithPath:` (page 127)
`+ bundleWithURL:` (page 118)

Declared In

`NSBundle.h`

bundleWithURL:

Returns an `NSBundle` object that corresponds to the specified file URL.

```
+ (NSBundle *)bundleWithURL:(NSURL *)url
```

Parameters

url

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

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

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

Availability
Available in iOS 4.0 and later.

See Also
`+ bundleWithPath:` (page 117)
`+ bundleWithIdentifier:` (page 117)
`+ bundleForClass:` (page 116)

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 iOS 2.0 and later.

See Also
`+ bundleForClass:` (page 116)
`+ bundleWithPath:` (page 117)

Related Sample Code
AddMusic
GKRocket
GKTank
MoviePlayer
ScrollViewSuite

Declared In
`NSBundle.h`

pathForResource:ofType:inDirectory:

Returns the full pathname for the resource file identified by the specified 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 directory specified by *bundlePath*.

extension
If *extension* is an empty string or `nil`, the extension is assumed not to exist and the file is the first file 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 a Mac OS X 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 non-localized resource directory of the specified bundle. (In Mac OS X, this directory is typically called `Resources` but in iOS, it is the main bundle directory.) 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 iOS 2.0 and later.

See Also
- `localizedStringForKey:value:table:` (page 131)
- `pathForResource:ofType:` (page 133)
- `pathForResource:ofType:inDirectory:` (page 134)

+ pathsForResourcesOfType:inDirectory: (page 120)
- pathsForResourcesOfType:inDirectory: (page 135)
- URLForResource:withExtension:subdirectory: (page 143)

Declared In
NSBundle.h

pathsForResourcesOfType:inDirectory:

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

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

Parameters

extension
The file extension. If *extension* is an empty string or nil, the extension is assumed not to exist, all the files in *bundlePath* 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 if 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 of the specified bundle. (In Mac OS X, this directory is typically called Resources but in iOS, it is the main bundle directory.) 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 iOS 2.0 and later.

See Also
- localizedStringForKey:value:table: (page 131)
- pathForResource:ofType: (page 133)
- pathForResource:ofType:inDirectory: (page 134)

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 iOS 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 iOS 2.0 and later.

Declared In

`NSBundle.h`

URLForResource:withExtension:subdirectory:inBundleWithURL:

Creates and returns a file URL for the resource with the specified name and extension in the specified bundle.

```
+ (NSURL *)URLForResource:(NSString *)name withExtension:(NSString *)ext
    subdirectory:(NSString *)subpath inBundleWithURL:(NSURL *)bundleURL
```

Parameters

name

The name of the resource file.

extension

If *extension* is an empty string or `nil`, the extension is assumed not to exist and the file URL is the first file encountered that exactly matches *name*.

subpath

The name of the bundle subdirectory to search.

bundleURL

The file URL of the bundle to search.

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

Availability
Available in iOS 4.0 and later.

Declared In
`NSBundle.h`

URLsForResourcesWithExtension:subdirectory:inBundleWithURL:

Returns an array containing the file URLs for all bundle resources having the specified filename extension, residing in the specified resource subdirectory, within the specified bundle.

```
+ (NSArray *)URLsForResourcesWithExtension:(NSString *)ext subdirectory:(NSString
    *)subpath inBundleWithURL:(NSURL *)bundleURL
```

Parameters

ext

The file extension of the files to retrieve.

subpath

The name of the bundle subdirectory to search.

bundleURL

The file URL of the bundle to search.

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

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Declared In
`NSBundle.h`

builtInPlugInsURL

Returns the file URL of the receiver's subdirectory containing plug-ins.

```
- (NSURL *)builtInPlugInsURL
```

Return Value
The file URL 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 URL for non-standard bundle formats or for some older bundle formats.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also
- `infoDictionary` (page 126)

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

Declared In
`NSBundle.h`

bundleURL

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

```
- (NSURL *)bundleURL
```

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

Availability
Available in iOS 4.0 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 147) after all classes and categories have been loaded; see "Notifications" (page 147) for details.

Availability
Available in iOS 2.0 and later.

See Also
- `principalClass` (page 138)
- `load` (page 128)

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 iOS 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 146). 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 146).

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
`NSBundle.h`

executableURL

Returns the file URL of the receiver's executable file.

```
- (NSURL *)executableURL
```

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

Availability
Available in iOS 4.0 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. The `NSBundle` class may add extra keys to this dictionary for its own use.

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

Availability
Available in iOS 2.0 and later.

See Also
- `principalClass` (page 138)
+ `dictionaryWithContentsOfFile:` (page 390) (NSDictionary)

Declared In
`NSBundle.h`

initWithPath:

Returns an `NSBundle` object initialized to correspond to the specified 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 118) class method to get this instance. You can also use the `bundleWithPath:` (page 117) class method to obtain a bundle identified by its directory path.

Availability
Available in iOS 2.0 and later.

See Also
+ `bundleForClass:` (page 116)
- `initWithURL:` (page 127)

Declared In
`NSBundle.h`

initWithURL:

Returns an `NSBundle` object initialized to correspond to the specified file URL.

```
- (id)initWithURL:(NSURL *)url
```

Parameters
url
The file URL to a directory. This must be a full URL for a directory; if it contains any symbolic links, they must be resolvable.

Return Value
An `NSBundle` object initialized to correspond to *`url`*. This method initializes and returns a new instance only if there is no existing bundle associated with *`url`*, otherwise it deallocates `self` and returns the existing object. If *`url`* 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 118) class method to get this instance. You can also use the `bundleWithURL:` (page 118) class method to obtain a bundle identified by its file URL.

Availability
Available in iOS 4.0 and later.

See Also
`+ bundleWithPath:` (page 117)
`+ bundleWithIdentifier:` (page 117)
`+ bundleForClass:` (page 116)
`+ bundleWithURL:` (page 118)

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 iOS 2.0 and later.

See Also
`- load` (page 128)

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 142) method.

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

Availability
Available in iOS 2.0 and later.

See Also
- `loadAndReturnError:` (page 129)
- `isLoaded` (page 128)
- `unload` (page 142)
- `classNamed:` (page 124)
- `principalClass` (page 138)

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 iOS 2.0 and later.

See Also
- `load` (page 128)
- `unload` (page 142)

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 iOS 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 iOS 2.0 and later.

Declared In
NSBundle.h

localizedStringForKey:value:table:

Returns a localized version of the string designated by the specified key and residing in the specified 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 131) 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 iOS 2.0 and later.

See Also
- `pathForResource:ofType:` (page 133)
- `pathForResource:ofType:inDirectory:` (page 134)
- `pathsForResourcesOfType:inDirectory:` (page 135)
+ `pathForResource:ofType:inDirectory:` (page 119)
+ `pathsForResourcesOfType:inDirectory:` (page 120)

Declared In
`NSBundle.h`

objectForInfoDictionaryKey:

Returns the value associated with the specified key in the receiver's information 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 iOS 2.0 and later.

Declared In
`NSBundle.h`

pathForAuxiliaryExecutable:

Returns the full pathname of the executable with the specified 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 iOS 2.0 and later.

Declared In
NSBundle.h

pathForResource:ofType:

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

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

Parameters

name

The name of the resource file.

extension

If *extension* is an empty string or nil, the extension is assumed not to exist and the file is the first file encountered that exactly matches *name*.

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

Discussion
The method first looks for a matching resource file in the non-localized resource directory of the specified bundle. (In Mac OS X, this directory is typically called Resources but in iOS, it is the main bundle directory.) 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 plist within the bundle, and loads it into an NSDictionary.

```
NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];
if (commonDictionaryPath = [thisBundle pathForResource:@"CommonDictionary"
ofType:@"plist"])  {
    theDictionary = [[NSDictionary alloc]
initWithContentsOfFile:commonDictionaryPath];
    // when completed, it is the developer's responsibility to release
theDictionary
}
```

Availability
Available in iOS 2.0 and later.

See Also
- URLForResource:withExtension: (page 143)

Related Sample Code
AddMusic
GKTank
MoviePlayer
ScrollViewSuite

Declared In
NSBundle.h

pathForResource:ofType:inDirectory:

Returns the full pathname for the resource identified by the specified 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

If *extension* is an empty string or `nil`, all the files in *subpath* and its subdirectories are returned. If an extension is provided the subdirectories are not searched.

subpath

The name of the bundle subdirectory. Can be `nil`.

Return Value
An array of full pathnames for the *subpath* or `nil` if no files are located.

Discussion
If *subpath* is `nil`, this method searches the top-level nonlocalized resource directory and the top-level of any language-specific directories. (In Mac OS X, the top-level nonlocalized resource directory is typically called `Resources` but in iOS, it is the main bundle directory.) For example, suppose you have a Mac OS X application with a modern bundle and you 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.

Whether this method recurses through subdirectories is dependent on the *extension* parameter. If `nil` or an empty string it will recurse, otherwise, it does not. (The search order for the language-specific directories corresponds to the user's preferences.) For more details see Bundles and Localization.

Availability
Available in iOS 2.0 and later.

See Also
- `localizedStringForKey:value:table:` (page 131)
- `pathForResource:ofType:` (page 133)
- `pathsForResourcesOfType:inDirectory:` (page 135)
+ `pathForResource:ofType:inDirectory:` (page 119)

Declared In
`NSBundle.h`

pathForResource:ofType:inDirectory:forLocalization:

Returns the full pathname for the resource identified by the specified 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

If *extension* is an empty string or `nil`, the extension is assumed not to exist and the file is the first file encountered that exactly matches *name*.

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 134), 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 121) for how to determine what localizations are available.

Availability
Available in iOS 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. If *extension* is an empty string or `nil`, the extension is assumed not to exist, all the files in *subpath* are returned.

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 and the top-level of any language-specific directories. (In Mac OS X, the top-level nonlocalized resource directory is typically called `Resources` but in iOS, it is the main bundle directory.) For example, suppose you have a Mac OS X application with a modern bundle and you 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 iOS 2.0 and later.

See Also
- `localizedStringForKey:value:table:` (page 131)
- `pathForResource:ofType:` (page 133)
- `pathForResource:ofType:inDirectory:` (page 134)
+ `pathForResource:ofType:inDirectory:` (page 119)
+ `pathsForResourcesOfType:inDirectory:` (page 120)

Declared In
`NSBundle.h`

pathsForResourcesOfType:inDirectory:forLocalization:

Returns an array containing the file 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 135), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

Availability
Available in iOS 2.0 and later.

Declared In
`NSBundle.h`

preferredLocalizations

Returns an array of strings indicating the actual localizations contained in the receiver's bundle.

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

Return Value
An array of `NSString` objects, each of which identifies the a localization in the receiver's bundle. The languages are in the preferred order.

Availability
Available in iOS 2.0 and later.

See Also
`+ preferredLocalizationsFromArray:` (page 121)
`- localizations` (page 130)

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 129) method.

Availability
Available in iOS 2.0 and later.

See Also
- `loadAndReturnError:` (page 129)

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 126), 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 non-loadable 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 147) after all classes and categories have been loaded; see "Notifications" (page 147) for details.

The following method obtains a bundle by specifying its path (`bundleWithPath:` (page 117)), then loads the bundle with `principalClass` (page 138) 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 iOS 2.0 and later.

See Also
- `classNamed:` (page 124)
- `infoDictionary` (page 126)
- `load` (page 128)

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 iOS 2.0 and later.

Declared In
`NSBundle.h`

privateFrameworksURL

Returns the file URL of the receiver's subdirectory containing private frameworks.

```
- (NSURL *)privateFrameworksURL
```

Return Value
The file URL 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 URL for non-standard bundle formats or for some older bundle formats.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also
- `bundlePath` (page 124)

Declared In
`NSBundle.h`

resourceURL

Returns the file URL of the receiver's subdirectory containing resource files.

```
- (NSURL *)resourceURL
```

Return Value
The file URL of the receiver's subdirectory containing resource 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 iOS 4.0 and later.

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 iOS 2.0 and later.

Declared In
NSBundle.h

sharedFrameworksURL

Returns the file URL of the receiver's subdirectory containing shared frameworks.

```
- (NSURL *)sharedFrameworksURL
```

Return Value
The file URL 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 URL for non-standard bundle formats or for some older bundle formats.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Declared In
NSBundle.h

sharedSupportURL

Returns the file URL of the receiver's subdirectory containing shared support files.

```
- (NSURL *)sharedSupportURL
```

Return Value
The file URL 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 iOS 4.0 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 iOS 2.0 and later.

See Also
- `loadAndReturnError:` (page 129)
- `load` (page 128)

Declared In
`NSBundle.h`

URLForAuxiliaryExecutable:

Returns the file URL of the executable with the specified name in the receiver's bundle.

```
- (NSURL *)URLForAuxiliaryExecutable:(NSString *)executableName
```

Parameters

executableName
The name of an executable file.

Return Value
The file URL 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 URL for non-standard bundle formats or for some older bundle formats.

Availability
Available in iOS 4.0 and later.

Declared In
NSBundle.h

URLForResource:withExtension:

Returns the file URL for the resource identified by the specified name and file extension.

```
- (NSURL *)URLForResource:(NSString *)name withExtension:(NSString *)extension
```

Parameters

name

The name of the resource file.

extension

If *extension* is an empty string or nil, the extension is assumed not to exist and the file URL is the first file encountered that exactly matches *name*.

Return Value
The file URL 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 of the specified bundle. (In Mac OS X, this directory is typically called Resources but in iOS, it is the main bundle directory.) 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.

Availability
Available in iOS 4.0 and later.

Declared In
NSBundle.h

URLForResource:withExtension:subdirectory:

Returns the file URL for the resource file identified by the specified name and extension and residing in a given bundle directory.

```
- (NSURL *)URLForResource:(NSString *)name withExtension:(NSString *)extension
    subdirectory:(NSString *)subpath
```

Parameters

name

The name of a resource file contained in the directory specified by *bundleURL*.

extension

If *extension* is an empty string or `nil`, the extension is assumed not to exist and the file URL is the first file encountered that exactly matches *name*.

subpath

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

Return Value

The file URL 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 non-localized resource directory of the specified bundle. (In Mac OS X, this directory is typically called `Resources` but in iOS, it is the main bundle directory.) 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.

Availability

Available in iOS 4.0 and later.

See Also

`- pathForResource:ofType:inDirectory:` (page 134)

Declared In

`NSBundle.h`

URLForResource:withExtension:subdirectory:localization:

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

```
- (NSURL *)URLForResource:(NSString *)name withExtension:(NSString *)extension
    subdirectory:(NSString *)subpath localization:(NSString *)localizationName
```

Parameters

name

The name of the resource file.

extension

If *extension* is an empty string or `nil`, the extension is assumed not to exist and the file URL is the first file encountered that exactly matches *name*.

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 file URL for the resource file or `nil` if the file could not be located.

Discussion
This method is equivalent to `URLsForResourcesWithExtension:subdirectory:` (page 145), 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 121) for how to determine what localizations are available.

Availability
Available in iOS 4.0 and later.

Declared In
`NSBundle.h`

URLsForResourcesWithExtension:subdirectory:

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

```
- (NSArray *)URLsForResourcesWithExtension:(NSString *)extension
    subdirectory:(NSString *)subpath
```

Parameters

`name`
The name of the resource file.

`extension`
If *`extension`* is an empty string or `nil`, the extension is assumed not to exist and the file URL is the first file encountered that exactly matches *`name`*.

`subpath`
The name of the bundle subdirectory.

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

Discussion
If *`subpath`* is `nil`, this method searches the top-level non-localized resource directory and the top-level of any language-specific directories. (In Mac OS X, the top-level non-localized resource directory is typically called `Resources` but in iOS, it is the main bundle directory.) For example, suppose you have a Mac OS X application with a modern bundle and you 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 iOS 4.0 and later.

Declared In
`NSBundle.h`

URLsForResourcesWithExtension:subdirectory:localization:

Returns an array containing the file URLs 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 *)URLsForResourcesWithExtension:(NSString *)extensions
    subdirectory:(NSString *)subpath localization:(NSString *)localizationName
```

Parameters

ext

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 file URLs 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 `URLsForResourcesWithExtension:subdirectory:` (page 145), except that only nonlocalized resources and those in the language-specific `.lproj` directory specified by *localizationName* are searched.

Availability

Available in iOS 4.0 and later.

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 iOS 2.0 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitecturePPC`
Specifies the 32-bit PowerPC architecture.
Available in iOS 2.0 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitectureX86_64`
Specifies the 64-bit Intel architecture.
Available in iOS 2.0 and later.
Declared in `NSBundle.h`.

`NSBundleExecutableArchitecturePPC64`
Specifies the 64-bit PowerPC architecture.
Available in iOS 2.0 and later.
Declared in `NSBundle.h`.

NSLoadedClasses

This constant is provided in the `userInfo` (page 841) dictionary of the `NSBundleDidLoadNotification` (page 147) notification.

```
NSString * const NSLoadedClasses;
```

Constants

`NSLoadedClasses`
An `NSArray` object containing the names (as `NSString` objects) of each class that was loaded
Available in iOS 2.0 and later.
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 124) or `principalClass` (page 138)), 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 an `NSLoadedClasses` (page 147) key.

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 iOS 2.0 and later.

Declared In
`NSBundle.h`

NSCache Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	NSCache.h

Overview

An `NSCache` object is a collection-like container, or cache, that stores key-value pairs, similar to the `NSDictionary` class. Developers often incorporate caches to temporarily store objects with transient data that are expensive to create. Reusing these objects can provide performance benefits, because their values do not have to be recalculated. However, the objects are not critical to the application and can be discarded if memory is tight. If discarded, their values will have to be recomputed again when needed.

While a key-value pair is in the cache, the cache maintains a strong reference to it if garbage collection is in effect; in memory-managed code, the cache retains the item. A common data type stored in `NSCache` objects is an object that implements the `NSDiscardableContent` protocol. Storing this type of object in a cache has benefits, because its content can be discarded when it is not needed anymore, thus saving memory. By default, `NSDiscardableContent` objects in the cache are automatically removed from the cache if their content is discarded, although this automatic removal policy can be changed. If an `NSDiscardableContent` object is put into the cache, the cache calls `discardContentIfPossible` (page 1563) on it upon its removal.

`NSCache` objects differ from other mutable collections in a few ways. First, the `NSCache` class incorporates various auto-removal policies, which ensure that it does not use too much of the system's memory. The system automatically carries out these policies if memory is needed by other applications. When invoked, these policies remove some items from the cache, minimizing its memory footprint. Second, you can add, remove, and query items in the cache from different threads without having to lock the cache yourself. Lastly, retrieving something from an `NSCache` object returns an autoreleased result. These features are necessary for the `NSCache` class, as the cache may decide to automatically mutate itself asynchronously behind the scenes if it is called to free up memory.

Tasks

Modifying the Cache Name

- `name` (page 152)
  Returns the name of the cache.
- `setName:` (page 155)
  Sets the cache's name attribute to a specific string.

Getting a Cached Value

- `objectForKey:` (page 153)
  Returns the value associated with a given key.

Adding and Removing Cached Values

- `setObject:forKey:` (page 155)
  Sets the value of the specified key in the cache.
- `setObject:forKey:cost:` (page 156)
  Sets the value of the specified key in the cache, and associates the key-value pair with the specified cost.
- `removeObjectForKey:` (page 153)
  Removes the value of the specified key in the cache.
- `removeAllObjects` (page 153)
  Empties the cache.

Managing Cache Size

- `countLimit` (page 151)
  Returns the maximum number of objects that the cache can currently hold.
- `setCountLimit:` (page 154)
  Sets the maximum number of objects that the cache can hold.
- `totalCostLimit` (page 157)
  Returns the maximum total cost that the cache can have before it starts evicting objects.
- `setTotalCostLimit:` (page 156)
  Sets the maximum total cost that the cache can have before it starts evicting objects.

Managing Discardable Content

- `evictsObjectsWithDiscardedContent` (page 152)
  Returns whether or not the cache will automatically evict discardable-content objects whose content has been discarded.

- `setEvictsObjectsWithDiscardedContent:` (page 155)
  Sets whether the cache will automatically evict `NSDiscardableContent` objects after the object's content has been discarded.

Managing the Delegate

- `delegate` (page 151)
  Returns the cache's delegate.
- `setDelegate:` (page 154)
  Makes the given object the cache's delegate.

Instance Methods

countLimit

Returns the maximum number of objects that the cache can currently hold.

```
- (NSUInteger)countLimit
```

Return Value
The maximum number of objects that the cache can currently hold.

Discussion
By default, `countLimit` will be set to `0`. Any `countLimit` less than or equal to `0` has no effect on the number of allowed entries in the cache. This limit is not a strict limit, and if the cache goes over the limit, an object in the cache could be evicted instantly, later, or possibly never, all depending on the implementation details of the cache.

Availability
Available in iOS 4.0 and later.

See Also
`- setCountLimit:` (page 154)

Declared In
`NSCache.h`

delegate

Returns the cache's delegate.

```
- (id)delegate
```

Return Value
The application delegate object.

Discussion
The delegate object is expected to conform to the `NSCacheDelegate` protocol.

Availability
Available in iOS 4.0 and later.

See Also
- `setDelegate:` (page 154)

Declared In
`NSCache.h`

evictsObjectsWithDiscardedContent

Returns whether or not the cache will automatically evict discardable-content objects whose content has been discarded.

```
- (BOOL)evictsObjectsWithDiscardedContent
```

Return Value
`YES` if the cache will evict the object after it is discarded; otherwise, `NO`.

Discussion
By default, `evictsObjectsWithDiscardedContent` is set to `YES`.

Availability
Available in iOS 4.0 and later.

See Also
- `setEvictsObjectsWithDiscardedContent:` (page 155)

Declared In
`NSCache.h`

name

Returns the name of the cache.

```
- (NSString *)name
```

Return Value
The name of the cache.

Discussion
Returns the empty string if no name is specified.

Availability
Available in iOS 4.0 and later.

See Also
- `setName:` (page 155)

Declared In
`NSCache.h`

objectForKey:

Returns the value associated with a given key.

```
- (id)objectForKey:(id)key
```

Parameters

key

An object identifying the value.

Return Value

The value associated with *key*, or NULL if no value is associated with *key*. The caller does not have to release the value returned to it.

Availability

Available in iOS 4.0 and later.

See Also

- `setObject:forKey:` (page 155)
- `setObject:forKey:cost:` (page 156)
- `removeObjectForKey:` (page 153)

Declared In

`NSCache.h`

removeAllObjects

Empties the cache.

```
- (void)removeAllObjects
```

Availability

Available in iOS 4.0 and later.

See Also

- `removeObjectForKey:` (page 153)

Declared In

`NSCache.h`

removeObjectForKey:

Removes the value of the specified key in the cache.

```
- (void)removeObjectForKey:(id)key
```

Parameters

key

The key identifying the value to be removed.

Availability

Available in iOS 4.0 and later.

See Also
- `removeAllObjects` (page 153)

Declared In
`NSCache.h`

setCountLimit:

Sets the maximum number of objects that the cache can hold.

```
- (void)setCountLimit:(NSUInteger)lim
```

Parameters

lim

The maximum number of objects that the cache will be allowed to hold.

Discussion
Setting the count limit to a number less than or equal to `0` will have no effect on the maximum size of the cache.

Availability
Available in iOS 4.0 and later.

See Also
- `countLimit` (page 151)

Declared In
`NSCache.h`

setDelegate:

Makes the given object the cache's delegate.

```
- (void)setDelegate:(id)del
```

Parameters

del

The object to be registered as the delegate.

Discussion
The delegate object is expected to conform to the `NSCacheDelegate` protocol.

Availability
Available in iOS 4.0 and later.

See Also
- `delegate` (page 151)

Declared In
`NSCache.h`

setEvictsObjectsWithDiscardedContent:

Sets whether the cache will automatically evict `NSDiscardableContent` objects after the object's content has been discarded.

```
- (void)setEvictsObjectsWithDiscardedContent:(BOOL)b
```

Parameters

b

If `YES`, the cache evicts `NSDiscardableContent` objects after the object's contents has been discarded; if `NO` the cache does not evict these objects.

Availability

Available in iOS 4.0 and later.

See Also

`- evictsObjectsWithDiscardedContent` (page 152)

Declared In

`NSCache.h`

setName:

Sets the cache's name attribute to a specific string.

```
- (void)setName:(NSString *)cacheName
```

Parameters

cacheName

The new name for the cache.

Availability

Available in iOS 4.0 and later.

See Also

`- name` (page 152)

Declared In

`NSCache.h`

setObject:forKey:

Sets the value of the specified key in the cache.

```
- (void)setObject:(id)obj forKey:(id)key
```

Parameters

obj

The object to be stored in the cache.

key

The key with which to associate the value.

Availability

Available in iOS 4.0 and later.

See Also
- `setObject:forKey:cost:` (page 156)

Declared In
`NSCache.h`

setObject:forKey:cost:

Sets the value of the specified key in the cache, and associates the key-value pair with the specified cost.

```
- (void)setObject:(id)obj forKey:(id)key cost:(NSUInteger)num
```

Parameters

obj

The object to store in the cache.

key

The key with which to associate the value.

num

The cost with which to associate the key-value pair.

Discussion
The `cost` value is used to compute a sum encompassing the costs of all the objects in the cache. When memory is limited or when the total cost of the cache eclipses the maximum allowed total cost, the cache could begin an eviction process to remove some of its elements. However, this eviction process is not in a guaranteed order. As a consequence, if you try to manipulate the cost values to achieve some specific behavior, the consequences could be detrimental to your program. Typically, the obvious cost is the size of the value in bytes. If that information is not readily available, you should not go through the trouble of trying to compute it, as doing so will drive up the cost of using the cache. Pass in 0 for the cost value if you otherwise have nothing useful to pass, or simply use the `setObject:forKey:` method, which does not require a `cost` value to be passed in.

Availability
Available in iOS 4.0 and later.

See Also
- `setObject:forKey:` (page 155)
- `setTotalCostLimit:` (page 156)
- `totalCostLimit` (page 157)

Declared In
`NSCache.h`

setTotalCostLimit:

Sets the maximum total cost that the cache can have before it starts evicting objects.

```
- (void)setTotalCostLimit:(NSUInteger)lim
```

Parameters

lim

The maximum total cost that the cache can have before it starts evicting objects.

Availability
Available in iOS 4.0 and later.

See Also
- `totalCostLimit` (page 157)

Declared In
`NSCache.h`

totalCostLimit

Returns the maximum total cost that the cache can have before it starts evicting objects.

```
- (NSUInteger)totalCostLimit
```

Return Value
The current maximum cost that the cache can have before it starts evicting objects.

Discussion
The default value is 0, which means there is no limit on the size of the cache. If you add an object to the cache, you may pass in a specified cost for the object, such as the size in bytes of the object. If adding this object to the cache causes the cache's total cost to rise above `totalCostLimit`, the cache could automatically evict some of its objects until its total cost falls below `totalCostLimit`. The order in which the cache evicts objects is not guaranteed. This limit is not a strict limit, and if the cache goes over the limit, an object in the cache could be evicted instantly, at a later point in time, or possibly never, all depending on the implementation details of the cache.

Availability
Available in iOS 4.0 and later.

See Also
- `setTotalCostLimit:` (page 156)
- `setObject:forKey:cost:` (page 156)

Declared In
`NSCache.h`

NSCachedURLResponse Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLCache.h
Companion guide	URL Loading System Programming Guide

Overview

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 160)
  Initializes an `NSCachedURLResponse` object.
- `initWithResponse:data:userInfo:storagePolicy:` (page 161)
  Initializes an NSCachedURLResponse object.

Getting Cached URL Response Properties

- `data` (page 160)
  Returns the receiver's cached data.
- `response` (page 161)
  Returns the `NSURLResponse` object associated with the receiver.

- `storagePolicy` (page 161)
  Returns the receiver's cache storage policy.
- `userInfo` (page 162)
  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

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

See Also
`- initWithResponse:data:userInfo:storagePolicy:` (page 161)

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

See Also
`- initWithResponse:data:` (page 160)

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

Declared In
`NSURLCache.h`

storagePolicy

Returns the receiver's cache storage policy.

```
- (NSURLCacheStoragePolicy)storagePolicy
```

Return Value
The receiver's cache storage policy.

Availability

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

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: iOS ignores this cache policy, and instead treats it as NSURLCacheStorageAllowedInMemoryOnly.

Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSURLCache.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSURLCache.h`

NSCalendar Class Reference

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

Overview

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 171), `dateByAddingComponents:toDate:options:` (page 171), 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 169) and

`components:fromDate:toDate:options:` (page 169)) 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 179)).

`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 168)
Returns the logical calendar for the current user.

`+ autoupdatingCurrentCalendar` (page 167)
Returns the current logical calendar for the current user.

Initializing a Calendar

`- initWithCalendarIdentifier:` (page 173)
Initializes a newly-allocated `NSCalendar` object for the calendar specified by a given identifier.

`- setFirstWeekday:` (page 177)
Sets the index of the first weekday for the receiver.

`- setLocale:` (page 177)
Sets the locale for the receiver.

`- setMinimumDaysInFirstWeek:` (page 177)
Sets the minimum number of days in the first week of the receiver.

`- setTimeZone:` (page 178)
Sets the time zone for the receiver.

Getting Information About a Calendar

`- calendarIdentifier` (page 168)
Returns the identifier for the receiver.

`- firstWeekday` (page 172)
Returns the index of the first weekday of the receiver.

`- locale` (page 173)
Returns the locale for the receiver.

`- maximumRangeOfUnit:` (page 173)
The maximum range limits of the values that a given unit can take on in the receive

- `minimumDaysInFirstWeek` (page 174)
  Returns the minimum number of days in the first week of the receiver.
- `minimumRangeOfUnit:` (page 174)
  Returns the minimum range limits of the values that a given unit can take on in the receiver.
- `ordinalityOfUnit:inUnit:forDate:` (page 175)
  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 175)
  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 176)
  Returns by reference the starting time and duration of a given calendar unit that contains a given date.
- `timeZone` (page 178)
  Returns the time zone for the receiver.

Calendrical Calculations

- `components:fromDate:` (page 169)
  Returns a `NSDateComponents` object containing a given date decomposed into specified components.
- `components:fromDate:toDate:options:` (page 169)
  Returns, as an `NSDateComponents` object using specified components, the difference between two supplied dates.
- `dateByAddingComponents:toDate:options:` (page 171)
  Returns a new `NSDate` object representing the absolute time calculated by adding given components to a given date.
- `dateFromComponents:` (page 171)
  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 168)).

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 iOS 2.0 and later.

See Also
`+ currentCalendar` (page 168)
`- initWithCalendarIdentifier:` (page 173)
`- calendarIdentifier` (page 168)

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 167)).

Availability
Available in iOS 2.0 and later.

See Also
`+ autoupdatingCurrentCalendar` (page 167)
`- initWithCalendarIdentifier:` (page 173)
`- calendarIdentifier` (page 168)

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 iOS 2.0 and later.

See Also
+ autoupdatingCurrentCalendar (page 167)
- initWithCalendarIdentifier: (page 173)

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 iOS 2.0 and later.

See Also
- dateFromComponents: (page 171)
- components:fromDate:toDate:options: (page 169)
- dateByAddingComponents:toDate:options: (page 171)

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 iOS 2.0 and later.

See Also

- `dateByAddingComponents:toDate:options:` (page 171)
- `dateFromComponents:` (page 171)

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 181) 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 iOS 2.0 and later.

See Also

- `dateFromComponents:` (page 171)
- `components:fromDate:toDate:options:` (page 169)

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 iOS 2.0 and later.

See Also

- `components:fromDate:` (page 169)
- `dateFromComponents:` (page 171)

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 iOS 2.0 and later.

See Also
- `setFirstWeekday:` (page 177)

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 iOS 2.0 and later.

See Also
- `+ autoupdatingCurrentCalendar` (page 167)
- `- calendarIdentifier` (page 168)

Declared In
`NSCalendar.h`

locale

Returns the locale for the receiver.

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

Return Value
The locale for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setLocale:` (page 177)

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 iOS 2.0 and later.

See Also

- minimumRangeOfUnit: (page 174)

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 iOS 2.0 and later.

See Also

- setMinimumDaysInFirstWeek: (page 177)

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 iOS 2.0 and later.

See Also
- `maximumRangeOfUnit:` (page 173)

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. Typically 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`. An exception is the week-in-month calculation, which returns `0` for days before the first week in the month containing the date.

Note that some computations can take a relatively long time.

Availability
Available in iOS 2.0 and later.

See Also
- `rangeOfUnit:inUnit:forDate:` (page 175)
- `rangeOfUnit:startDate:interval:forDate:` (page 176)

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 iOS 2.0 and later.

See Also

- rangeOfUnit:startDate:interval:forDate: (page 176)
- ordinalityOfUnit:inUnit:forDate: (page 175)

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 179) 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 iOS 2.0 and later.

See Also
- `rangeOfUnit:inUnit:forDate:` (page 175)
- `ordinalityOfUnit:inUnit:forDate:` (page 175)

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 iOS 2.0 and later.

See Also
- `firstWeekday` (page 172)

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 iOS 2.0 and later.

See Also
- `locale` (page 173)

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 iOS 2.0 and later.

See Also
- minimumDaysInFirstWeek (page 174)

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 iOS 2.0 and later.

See Also
- timeZone (page 178)

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 iOS 2.0 and later.

See Also
- setTimeZone: (page 178)

Declared In
NSCalendar.h

Constants

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
    NSQuarterCalendarUnit = kCFCalendarUnitQuarter,
};
typedef NSUInteger NSCalendarUnit;
```

Constants

`NSEraCalendarUnit`

Specifies the era unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSYearCalendarUnit`

Specifies the year unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSMonthCalendarUnit`

Specifies the month unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSDayCalendarUnit`

Specifies the day unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSHourCalendarUnit`
Specifies the hour unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSMinuteCalendarUnit`
Specifies the minute unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSSecondCalendarUnit`
Specifies the second unit.

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

Available in iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSWeekCalendarUnit`
Specifies the week unit.

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

Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSCalendar.h`.

`NSQuarterCalendarUnit`
Specifies the quarter of the calendar as an `int`. Equal to `kCFCalendarUnitQuarter`.

Available in iOS 4.0 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSCharacterSet.h
Companion guide	String Programming Guide

Overview

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 195), provides the basis for all other instance methods in its interface. A subclass of `NSCharacterSet` needs only to implement this method, plus `mutableCopyWithZone:` (page 1614), for proper behavior. For optimal performance, a subclass should also override `bitmapRepresentation` (page 194), which otherwise works by invoking `characterIsMember:` (page 195) for every possible Unicode value.

`NSCharacterSet` is "toll-free bridged" with its Cocoa Foundation counterpart, *CFCharacterSet 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 `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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating a Standard Character Set

`+ alphanumericCharacterSet` (page 186)
Returns a character set containing the characters in the categories Letters, Marks, and Numbers.

`+ capitalizedLetterCharacterSet` (page 186)
Returns a character set containing the characters in the category of Titlecase Letters.

`+ controlCharacterSet` (page 189)
Returns a character set containing the characters in the categories of Control or Format Characters.

`+ decimalDigitCharacterSet` (page 189)
Returns a character set containing the characters in the category of Decimal Numbers.

`+ decomposableCharacterSet` (page 190)
Returns a character set containing all individual Unicode characters that can also be represented as composed character sequences.

`+ illegalCharacterSet` (page 190)
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 190)
Returns a character set containing the characters in the categories Letters and Marks.

`+ lowercaseLetterCharacterSet` (page 191)
Returns a character set containing the characters in the category of Lowercase Letters.

`+ newlineCharacterSet` (page 191)
Returns a character set containing the newline characters.

+ nonBaseCharacterSet (page 192)
Returns a character set containing the characters in the category of Marks.

+ punctuationCharacterSet (page 192)
Returns a character set containing the characters in the category of Punctuation.

+ symbolCharacterSet (page 192)
Returns a character set containing the characters in the category of Symbols.

+ uppercaseLetterCharacterSet (page 193)
Returns a character set containing the characters in the categories of Uppercase Letters and Titlecase Letters.

+ whitespaceAndNewlineCharacterSet (page 193)
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 194)
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 187)
Returns a character set containing the characters in a given string.

+ characterSetWithRange: (page 188)
Returns a character set containing characters with Unicode values in a given range.

- invertedSet (page 196)
Returns a character set containing only characters that don't exist in the receiver.

Creating and Managing Character Sets as Bitmap Representations

+ characterSetWithBitmapRepresentation: (page 187)
Returns a character set containing characters determined by a given bitmap representation.

+ characterSetWithContentsOfFile: (page 188)
Returns a character set read from the bitmap representation stored in the file a given path.

- bitmapRepresentation (page 194)
Returns an NSData object encoding the receiver in binary format.

Testing Set Membership

- characterIsMember: (page 195)
Returns a Boolean value that indicates whether a given character is in the receiver.

- hasMemberInPlane: (page 195)
Returns a Boolean value that indicates whether the receiver has at least one member in a given character plane.

- isSupersetOfSet: (page 196)
Returns a Boolean value that indicates whether the receiver is a superset of another given character set.

- longCharacterIsMember: (page 196)
  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 iOS 2.0 and later.

See Also
+ letterCharacterSet (page 190)
+ decimalDigitCharacterSet (page 189)

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 iOS 2.0 and later.

See Also
+ letterCharacterSet (page 190)
+ uppercaseLetterCharacterSet (page 193)

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 iOS 2.0 and later.

See Also

- `bitmapRepresentation` (page 194)
+ `characterSetWithContentsOfFile:` (page 188)

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 iOS 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 261) and pass the result to `characterSetWithBitmapRepresentation:` (page 187).

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 iOS 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 return; *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 iOS 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 iOS 2.0 and later.

See Also
+ illegalCharacterSet (page 190)

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 iOS 2.0 and later.

See Also
+ alphanumericCharacterSet (page 186)

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 iOS 2.0 and later.

See Also
`+ nonBaseCharacterSet` (page 192)

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 iOS 2.0 and later.

See Also
`+ controlCharacterSet` (page 189)

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 iOS 2.0 and later.

See Also
`+ alphanumericCharacterSet` (page 186)
`+ lowercaseLetterCharacterSet` (page 191)
`+ uppercaseLetterCharacterSet` (page 193)

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 iOS 2.0 and later.

See Also
`+ uppercaseLetterCharacterSet` (page 193)
`+ letterCharacterSet` (page 190)

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 iOS 2.0 and later.

See Also
`+ whitespaceAndNewlineCharacterSet` (page 193)

+ whitespaceCharacterSet (page 194)

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 iOS 2.0 and later.

See Also
+ decomposableCharacterSet (page 190)

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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
+ capitalizedLetterCharacterSet (page 186)
+ lowercaseLetterCharacterSet (page 191)
+ letterCharacterSet (page 190)

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 iOS 2.0 and later.

See Also
+ newlineCharacterSet (page 191)
+ whitespaceCharacterSet (page 194)

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 iOS 2.0 and later.

See Also
+ whitespaceAndNewlineCharacterSet (page 193)
+ newlineCharacterSet (page 191)

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 iOS 2.0 and later.

See Also
`+ characterSetWithBitmapRepresentation:` (page 187)

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 iOS 2.0 and later.

See Also
`- longCharacterIsMember:` (page 196)

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 iOS 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 iOS 2.0 and later.

See Also
`invert` (page 762) (`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 iOS 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 iOS 2.0 and later.

See Also
`- characterIsMember:` (page 195)

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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSCoder.h Foundation/NSKeyedArchiver.h Foundation/NSGeometry.h
Companion guide	Archives and Serializations Programming Guide
Related sample code	aurioTouch GLSprite SpeakHere

Overview

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*.

Tasks

Testing Coder

- `allowsKeyedCoding` (page 202)
  Returns a Boolean value that indicates whether the receiver supports keyed coding of objects.
- `containsValueForKey:` (page 203)
  Returns a Boolean value that indicates whether an encoded value is available for a string.

Encoding Data

- `encodeArrayOfObjCType:count:at:` (page 209)
  Encodes an array of *count* items, whose Objective-C type is given by *itemType*.
- `encodeBool:forKey:` (page 210)
  Encodes *boolv* and associates it with the string *key*.
- `encodeBycopyObject:` (page 210)
  Can be overridden by subclasses to encode *object* so that a copy, rather than a proxy, is created upon decoding.
- `encodeByrefObject:` (page 210)
  Can be overridden by subclasses to encode *object* so that a proxy, rather than a copy, is created upon decoding.
- `encodeBytes:length:` (page 211)
  Encodes a buffer of data whose types are unspecified.
- `encodeBytes:length:forKey:` (page 211)
  Encodes a buffer of data, *bytesp*, whose length is specified by *lenv*, and associates it with the string *key*.
- `encodeConditionalObject:` (page 212)
  Can be overridden by subclasses to conditionally encode *object*, preserving common references to that object.
- `encodeConditionalObject:forKey:` (page 212)
  Conditionally encodes a reference to *objv* and associates it with the string *key* only if *objv* has been unconditionally encoded with `encodeObject:forKey:` (page 216).
- `encodeDataObject:` (page 212)
  Encodes a given `NSData` object.
- `encodeDouble:forKey:` (page 213)
  Encodes *realv* and associates it with the string *key*.
- `encodeFloat:forKey:` (page 213)
  Encodes *realv* and associates it with the string *key*.
- `encodeInt:forKey:` (page 214)
  Encodes *intv* and associates it with the string *key*.
- `encodeInteger:forKey:` (page 215)
  Encodes a given `NSInteger` and associates it with a given key.

- `encodeInt32:forKey:` (page 214)
  Encodes the 32-bit integer *intv* and associates it with the string *key*.
- `encodeInt64:forKey:` (page 214)
  Encodes the 64-bit integer *intv* and associates it with the string *key*.
- `encodeObject:` (page 215)
  Encodes *object*.
- `encodeObject:forKey:` (page 216)
  Encodes the object *objv* and associates it with the string *key*.
- `encodeRootObject:` (page 216)
  Can be overridden by subclasses to encode an interconnected group of Objective-C objects, starting with *rootObject*.
- `encodeValueOfObjCType:at:` (page 216)
  Must be overridden by subclasses to encode a single value residing at *address*, whose Objective-C type is given by *valueType*.
- `encodeValuesOfObjCTypes:` (page 217)
  Encodes a series of values of potentially differing Objective-C types.

Decoding Data

- `decodeArrayOfObjCType:count:at:` (page 203)
  Decodes an array of *count* items, whose Objective-C type is given by *itemType*.
- `decodeBoolForKey:` (page 204)
  Decodes and returns a boolean value that was previously encoded with `encodeBool:forKey:` (page 210) and associated with the string *key*.
- `decodeBytesForKey:returnedLength:` (page 204)
  Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 211) and associated with the string *key*.
- `decodeBytesWithReturnedLength:` (page 204)
  Decodes a buffer of data whose types are unspecified.
- `decodeDataObject` (page 205)
  Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 212). Subclasses must override this method.
- `decodeDoubleForKey:` (page 205)
  Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 213) or `encodeDouble:forKey:` (page 213) and associated with the string *key*.
- `decodeFloatForKey:` (page 205)
  Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 213) or `encodeDouble:forKey:` (page 213) and associated with the string *key*.
- `decodeIntForKey:` (page 207)
  Decodes and returns an `int` value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) and associated with the string *key*.

- `decodeIntegerForKey:` (page 206)
  Decodes and returns an `NSInteger` value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) and associated with the string *key*.
- `decodeInt32ForKey:` (page 206)
  Decodes and returns a 32-bit integer value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) and associated with the string *key*.
- `decodeInt64ForKey:` (page 206)
  Decodes and returns a 64-bit integer value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) and associated with the string *key*.
- `decodeObject` (page 207)
  Decodes an Objective-C object that was previously encoded with any of the `encode...Object:` methods.
- `decodeObjectForKey:` (page 208)
  Decodes and returns an autoreleased Objective-C object that was previously encoded with `encodeObject:forKey:` (page 216) or `encodeConditionalObject:forKey:` (page 212) and associated with the string *key*.
- `decodeValueOfObjCType:at:` (page 208)
  Decodes a single value, whose Objective-C type is given by *valueType*.
- `decodeValuesOfObjCTypes:` (page 209)
  Decodes a series of potentially different Objective-C types.

Managing Zones

- `objectZone` (page 218)
  Returns the memory zone used to allocate decoded objects.
- `setObjectZone:` (page 218)
  `NSCoder`'s implementation of this method does nothing.

Getting Version Information

- `systemVersion` (page 218)
  During encoding, this method should return the system version currently in effect.
- `versionForClassName:` (page 219)
  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 iOS 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 iOS 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 208) 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 209) 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 Programming Language*.

Availability
Available in iOS 2.0 and later.

See Also
`- decodeValuesOfObjCTypes:` (page 209)

Declared In
`NSCoder.h`

decodeBoolForKey:

Decodes and returns a boolean value that was previously encoded with `encodeBool:forKey:` (page 210) and associated with the string *key*.

```
- (BOOL)decodeBoolForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iOS 2.0 and later.

Declared In
`NSCoder.h`

decodeBytesForKey:returnedLength:

Decodes a buffer of data that was previously encoded with `encodeBytes:length:forKey:` (page 211) 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 iOS 2.0 and later.

See Also
- `encodeBytes:length:forKey:` (page 211)

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 208) 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 211) message used during encoding.

Availability
Available in iOS 2.0 and later.

See Also
- `encodeArrayOfObjCType:count:at:` (page 209)

Declared In
`NSCoder.h`

decodeDataObject

Decodes and returns an NSData object that was previously encoded with `encodeDataObject:` (page 212). Subclasses must override this method.

```
- (NSData *)decodeDataObject
```

Discussion
The implementation of your overriding method must match the implementation of your `encodeDataObject:` (page 212) method. For example, a typical `encodeDataObject:` (page 212) 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 iOS 2.0 and later.

Declared In
`NSCoder.h`

decodeDoubleForKey:

Decodes and returns a `double` value that was previously encoded with either `encodeFloat:forKey:` (page 213) or `encodeDouble:forKey:` (page 213) and associated with the string *key*.

```
- (double)decodeDoubleForKey:(NSString *)key
```

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iOS 2.0 and later.

Declared In
`NSCoder.h`

decodeFloatForKey:

Decodes and returns a `float` value that was previously encoded with `encodeFloat:forKey:` (page 213) or `encodeDouble:forKey:` (page 213) 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 iOS 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 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) 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 iOS 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 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) 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 iOS 2.0 and later.

Declared In
`NSCoder.h`

decodeIntegerForKey:

Decodes and returns an `NSInteger` value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) 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 iOS 2.0 and later.

Declared In
`NSCoder.h`

decodeIntForKey:

Decodes and returns an `int` value that was previously encoded with `encodeInt:forKey:` (page 214), `encodeInteger:forKey:` (page 215), `encodeInt32:forKey:` (page 214), or `encodeInt64:forKey:` (page 214) 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 iOS 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 208) 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 212) 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 iOS 2.0 and later.

See Also
- `encodeBycopyObject:` (page 210)

- encodeByrefObject: (page 210)
- encodeObject: (page 215)

Declared In
NSCoder.h

decodeObjectForKey:

Decodes and returns an autoreleased Objective-C object that was previously encoded with encodeObject:forKey: (page 216) or encodeConditionalObject:forKey: (page 212) and associated with the string *key*.

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

Discussion
Subclasses must override this method if they perform keyed coding.

Availability
Available in iOS 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 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 216) message used during encoding.

Availability
Available in iOS 2.0 and later.

See Also
- decodeArrayOfObjCType:count:at: (page 203)
- decodeValuesOfObjCTypes: (page 209)
- decodeObject (page 207)

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 217) message used during encoding.

`NSCoder`'s implementation invokes `decodeValueOfObjCType:at:` (page 208) to decode individual types. Subclasses that implement the `decodeValueOfObjCType:at:` (page 208) 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 Programming Language*.

Availability
Available in iOS 2.0 and later.

See Also
- `decodeArrayOfObjCType:count:at:` (page 203)

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 216) to encode the entire array of items. Subclasses that implement the `encodeValueOfObjCType:at:` (page 216) method do not need to override this method.

This method must be matched by a subsequent `decodeArrayOfObjCType:count:at:` (page 203) 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 iOS 2.0 and later.

See Also
- `encodeValueOfObjCType:at:` (page 216)
- `encodeValuesOfObjCTypes:` (page 217)

- encodeBytes:length: (page 211)

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 iOS 2.0 and later.

See Also
- decodeBoolForKey: (page 204)

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 215).

This method must be matched by a corresponding decodeObject (page 207) message.

Availability
Available in iOS 2.0 and later.

See Also
- encodeRootObject: (page 216)
- encodeConditionalObject: (page 212)
- encodeByrefObject: (page 210)

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 215).

This method must be matched by a corresponding `decodeObject` (page 207) message.

Availability
Available in iOS 2.0 and later.

See Also
`- encodeBycopyObject:` (page 210)

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 204) message.

Availability
Available in iOS 2.0 and later.

See Also
`- encodeArrayOfObjCType:count:at:` (page 209)

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 iOS 2.0 and later.

See Also
`- decodeBytesForKey:returnedLength:` (page 204)

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 207) 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 215).

Availability
Available in iOS 2.0 and later.

See Also
- `encodeRootObject:` (page 216)
- `encodeObject:` (page 215)
- `encodeBycopyObject:` (page 210)
- `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 216).

```
- (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 208) method. If *objv* was never encoded unconditionally, `decodeObjectForKey:` (page 208) returns `nil` in place of *objv*.

Availability
Available in iOS 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 205) message.

Availability
Available in iOS 2.0 and later.

See Also
- `encodeObject:` (page 215)

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 iOS 2.0 and later.

See Also
- `decodeDoubleForKey:` (page 205)
- `decodeFloatForKey:` (page 205)

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 iOS 2.0 and later.

See Also
- `decodeFloatForKey:` (page 205)
- `decodeDoubleForKey:` (page 205)

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 iOS 2.0 and later.

See Also
- `decodeIntForKey:` (page 207)
- `decodeIntegerForKey:` (page 206)
- `decodeInt32ForKey:` (page 206)
- `decodeInt64ForKey:` (page 206)

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 iOS 2.0 and later.

See Also
- `decodeIntForKey:` (page 207)
- `decodeIntegerForKey:` (page 206)
- `decodeInt32ForKey:` (page 206)
- `decodeInt64ForKey:` (page 206)

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 iOS 2.0 and later.

See Also
- `decodeIntForKey:` (page 207)
- `decodeIntegerForKey:` (page 206)
- `decodeInt32ForKey:` (page 206)
- `decodeInt64ForKey:` (page 206)

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 iOS 2.0 and later.

See Also
- `decodeIntForKey:` (page 207)
- `decodeIntegerForKey:` (page 206)
- `decodeInt32ForKey:` (page 206)
- `decodeInt64ForKey:` (page 206)

Declared In
`NSCoder.h`

encodeObject:

Encodes *`object`*.

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

Discussion
`NSCoder`'s implementation simply invokes `encodeValueOfObjCType:at:` (page 216) 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 207) message.

Availability
Available in iOS 2.0 and later.

See Also
- `encodeRootObject:` (page 216)
- `encodeConditionalObject:` (page 212)
- `encodeBycopyObject:` (page 210)

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 iOS 2.0 and later.

See Also
- decodeObjectForKey: (page 208)

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 215).

This method must be matched by a subsequent decodeObject (page 207) message.

Availability
Available in iOS 2.0 and later.

See Also
- encodeObject: (page 215)
- encodeConditionalObject: (page 212)
- encodeBycopyObject: (page 210)
- 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 208) 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 Programming Language*.

Availability
Available in iOS 2.0 and later.

See Also
- `encodeArrayOfObjCType:count:at:` (page 209)
- `encodeValuesOfObjCTypes:` (page 217)

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 209) message.

`NSCoder`'s implementation invokes `encodeValueOfObjCType:at:` (page 216) to encode individual types. Subclasses that implement the `encodeValueOfObjCType:at:` (page 216) 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 Programming Language*.

Availability
Available in iOS 2.0 and later.

See Also
- `encodeArrayOfObjCType:count:at:` (page 209)
- `encodeValueOfObjCType:at:` (page 216)

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 218) 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 iOS 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 218) 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
+ setVersion: (page 962) (NSObject)
+ version (page 962) (NSObject)

Declared In
NSCoder.h

NSComparisonPredicate Class Reference

Inherits from	NSPredicate : NSObject
Conforms to	NSCoding (NSPredicate) NSCopying (NSPredicate) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	Foundation/NSComparisonPredicate.h
Companion guide	Predicate Programming Guide

Overview

`NSComparisonPredicate` is a subclass of `NSPredicate` that you use to compare expressions.

You use comparison predicates to compare the results of two expressions. You create a comparison predicate with an operator, a left expression, and a right expression. You represent the expressions using instances of the `NSExpression` class. When you evaluate the predicate, it returns as a `BOOL` value the result of invoking the operator with the results of evaluating the expressions.

Tasks

Constructors

`+ predicateWithLeftExpression:rightExpression:customSelector:` (page 222)
Returns a new predicate formed by combining the left and right expressions using a given selector.

`+ predicateWithLeftExpression:rightExpression:modifier:type:options:` (page 222)
Creates and returns a predicate of a given type formed by combining given left and right expressions using a given modifier and options.

`- initWithLeftExpression:rightExpression:customSelector:` (page 224)
Initializes a predicate formed by combining given left and right expressions using a given selector.

`- initWithLeftExpression:rightExpression:modifier:type:options:` (page 224)
Initializes a predicate to a given type formed by combining given left and right expressions using a given modifier and options.

Getting Information About a Comparison Predicate

- `comparisonPredicateModifier` (page 223)
  Returns the comparison predicate modifier for the receiver.
- `customSelector` (page 223)
  Returns the selector for the receiver.
- `leftExpression` (page 225)
  Returns the left expression for the receiver.
- `options` (page 225)
  Returns the options that are set for the receiver.
- `predicateOperatorType` (page 225)
  Returns the predicate type for the receiver.
- `rightExpression` (page 226)
  Returns the right expression for the receiver.

Class Methods

predicateWithLeftExpression:rightExpression:customSelector:

Returns a new predicate formed by combining the left and right expressions using a given selector.

```
+ (NSPredicate *)predicateWithLeftExpression:(NSExpression *)lhs
    rightExpression:(NSExpression *)rhs customSelector:(SEL)selector
```

Parameters

lhs
The left hand side expression.

rhs
The right hand side expression.

selector
The selector to use for comparison. The method defined by the selector must take a single argument and return a `BOOL` value.

Return Value
A new predicate formed by combining the left and right expressions using `selector`.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

predicateWithLeftExpression:rightExpression:modifier:type:options:

Creates and returns a predicate of a given type formed by combining given left and right expressions using a given modifier and options.

```
+ (NSPredicate *)predicateWithLeftExpression:(NSExpression *)lhs
    rightExpression:(NSExpression *)rhs
    modifier:(NSComparisonPredicateModifier)modifier
    type:(NSPredicateOperatorType)type options:(NSUInteger)options
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

modifier
The modifier to apply.

type
The predicate operator type.

options
The options to apply (see "NSComparisonPredicate Options" (page 227)). For no options, pass `0`.

Return Value
A new predicate of type *type* formed by combining the given left and right expressions using the *modifier* and *options*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

Instance Methods

comparisonPredicateModifier

Returns the comparison predicate modifier for the receiver.

```
- (NSComparisonPredicateModifier)comparisonPredicateModifier
```

Return Value
The comparison predicate modifier for the receiver.

Discussion
The default value is `NSDirectPredicateModifier` (page 226).

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

customSelector

Returns the selector for the receiver.

```
- (SEL)customSelector
```

Return Value
The selector for the receiver, or `NULL` if there is none.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

initWithLeftExpression:rightExpression:customSelector:

Initializes a predicate formed by combining given left and right expressions using a given selector.

```
- (id)initWithLeftExpression:(NSExpression *)lhs rightExpression:(NSExpression
    *)rhs customSelector:(SEL)selector
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

selector
The selector to use. The method defined by the selector must take a single argument and return a `BOOL` value.

Return Value
The receiver, initialized by combining the left and right expressions using *selector*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

initWithLeftExpression:rightExpression:modifier:type:options:

Initializes a predicate to a given type formed by combining given left and right expressions using a given modifier and options.

```
- (id)initWithLeftExpression:(NSExpression *)lhs rightExpression:(NSExpression
    *)rhs modifier:(NSComparisonPredicateModifier)modifier
    type:(NSPredicateOperatorType)type options:(NSUInteger)options
```

Parameters

lhs
The left hand expression.

rhs
The right hand expression.

modifier
The modifier to apply.

type
The predicate operator type.

options
The options to apply (see "NSComparisonPredicate Options" (page 227)). For no options, pass `0`.

Return Value
The receiver, initialized to a predicate of type *type* formed by combining the left and right expressions using the *modifier* and *options*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

leftExpression

Returns the left expression for the receiver.

```
- (NSExpression *)leftExpression
```

Return Value
The left expression for the receiver, or `nil` if there is none.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

options

Returns the options that are set for the receiver.

```
- (NSUInteger)options
```

Return Value
The options that are set for the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

predicateOperatorType

Returns the predicate type for the receiver.

```
- (NSPredicateOperatorType)predicateOperatorType
```

Return Value
The predicate type for the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

rightExpression

Returns the right expression for the receiver.

```
- (NSExpression *)rightExpression
```

Return Value
The right expression for the receiver, or `nil` if there is none.

Availability
Available in iOS 3.0 and later.

Declared In
`NSComparisonPredicate.h`

Constants

NSComparisonPredicateModifier

These constants describe the possible types of modifier for `NSComparisonPredicate`.

```
typedef enum {
    NSDirectPredicateModifier = 0,
    NSAllPredicateModifier,
    NSAnyPredicateModifier,
} NSComparisonPredicateModifier;
```

Constants

`NSDirectPredicateModifier`

A predicate to compare directly the left and right hand sides.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSAllPredicateModifier`

A predicate to compare all entries in the destination of a to-many relationship.

The left hand side must be a collection. The corresponding predicate compares each value in the left hand side with the right hand side, and returns `NO` when it finds the first mismatch—`YES` if all match.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSAnyPredicateModifier`

A predicate to match with any entry in the destination of a to-many relationship.

The left hand side must be a collection. The corresponding predicate compares each value in the left hand side against the right hand side and returns `YES` when it finds the first match—or `NO` if no match is found

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

NSComparisonPredicate Options

These constants describe the possible types of string comparison for `NSComparisonPredicate`. These options are supported for `LIKE` as well as all of the equality/comparison operators.

```
enum {
    NSCaseInsensitivePredicateOption = 0x01,
    NSDiacriticInsensitivePredicateOption = 0x02,
    NSNormalizedPredicateOption = 0x04,
    NSLocaleSensitivePredicateOption = 0x08
};
```

Constants

`NSCaseInsensitivePredicateOption`

A case-insensitive predicate.

You represent this option in a predicate format string using a `[c]` following a string operation (for example, `"NeXT" like[c] "next"`).

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSDiacriticInsensitivePredicateOption`

A diacritic-insensitive predicate.

You represent this option in a predicate format string using a `[d]` following a string operation (for example, `"naïve" like[d] "naive"`).

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSNormalizedPredicateOption`

Indicates that the strings to be compared have been preprocessed.

This option supersedes `NSCaseInsensitivePredicateOption` and `NSDiacriticInsensitivePredicateOption`, and is intended as a performance optimization option.

You represent this option in a predicate format string using a `[n]` following a string operation (for example, `"WXYZlan" matches[n] ".lan"`).

`NSLocaleSensitivePredicateOption`

Indicates that strings to be compared using <, <=, =, =>, > should be handled in a locale-aware fashion.

You represent this option in a predicate format string using a `[l]` following one of the <, <=, =, =>, > operators (for example, `"straße" >[l] "strasse"`).

NSPredicateOperatorType

Defines the type of comparison for `NSComparisonPredicate`.

```
typedef enum {
    NSLessThanPredicateOperatorType = 0,
    NSLessThanOrEqualToPredicateOperatorType,
    NSGreaterThanPredicateOperatorType,
    NSGreaterThanOrEqualToPredicateOperatorType,
    NSEqualToPredicateOperatorType,
    NSNotEqualToPredicateOperatorType,
    NSMatchesPredicateOperatorType,
    NSLikePredicateOperatorType,
    NSBeginsWithPredicateOperatorType,
    NSEndsWithPredicateOperatorType,
    NSInPredicateOperatorType,
    NSCustomSelectorPredicateOperatorType,
    NSContainsPredicateOperatorType,
    NSBetweenPredicateOperatorType
} NSPredicateOperatorType;
```

Constants

`NSLessThanPredicateOperatorType`
A less-than predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSLessThanOrEqualToPredicateOperatorType`
A less-than-or-equal-to predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSGreaterThanPredicateOperatorType`
A greater-than predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSGreaterThanOrEqualToPredicateOperatorType`
A greater-than-or-equal-to predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSEqualToPredicateOperatorType`
An equal-to predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSNotEqualToPredicateOperatorType`
A not-equal-to predicate.
Available in iOS 3.0 and later.
Declared in `NSComparisonPredicate.h`.

`NSMatchesPredicateOperatorType`
A full regular expression matching predicate.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSLikePredicateOperatorType`
A simple subset of the MATCHES predicate, similar in behavior to SQL `LIKE`.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSBeginsWithPredicateOperatorType`
A begins-with predicate.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSEndsWithPredicateOperatorType`
An ends-with predicate.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSInPredicateOperatorType`
A predicate to determine if the left hand side is in the right hand side.

For strings, returns `YES` if the left hand side is a substring of the right hand side . For collections, returns `YES` if the left hand side is in the right hand side .

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSCustomSelectorPredicateOperatorType`
A predicate that uses a custom selector that takes a single argument and returns a `BOOL` value.

The selector is invoked on the left hand side with the right hand side as the argument.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSContainsPredicateOperatorType`
A predicate to determine if the left hand side contains the right hand side.

Returns `YES` if `[lhs contains rhs]`; the left hand side must be an `NSExpression` object that evaluates to a collection

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

`NSBetweenPredicateOperatorType`
A predicate to determine if the right hand side lies at or between bounds specified by the left hand side.

Returns `YES` if `[lhs between rhs]`; the right hand side must be an array in which the first element sets the lower bound and the second element the upper, inclusive. Comparison is performed using compare: or the class-appropriate equivalent.

Available in iOS 3.0 and later.

Declared in `NSComparisonPredicate.h`.

NSCompoundPredicate Class Reference

Inherits from	NSPredicate : NSObject
Conforms to	NSCoding (NSPredicate) NSCopying (NSPredicate) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	Foundation/NSCompoundPredicate.h
Companion guide	Predicate Programming Guide

Overview

`NSCompoundPredicate` is a subclass of `NSPredicate` used to represent logical "gate" operations (AND/OR/NOT) and comparison operations.

Comparison operations are based on two expressions, as represented by instances of the `NSExpression` class. Expressions are created for constant values, key paths, and so on.

In Mac OS X v10.5 and later and in iOS, you can use `NSCompoundPredicate` to create an AND or OR compound predicate (but not a NOT compound predicate) using an array with 0, 1, or more elements:

- An AND predicate with no subpredicates evaluates to TRUE.
- An OR predicate with no subpredicates evaluates to FALSE.
- A compound predicate with one or more subpredicates evaluates to the truth of its subpredicates.

Tasks

Constructors

`+ andPredicateWithSubpredicates:` (page 232)
Returns a new predicate formed by AND-ing the predicates in a given array.

`+ notPredicateWithSubpredicate:` (page 232)
Returns a new predicate formed by NOT-ing a given predicate.

`+ orPredicateWithSubpredicates:` (page 233)
Returns a new predicate formed by OR-ing the predicates in a given array.

`- initWithType:subpredicates:` (page 234)
Returns the receiver initialized to a given type using predicates from a given array.

Getting Information About a Compound Predicate

`- compoundPredicateType` (page 233)
Returns the predicate type for the receiver.

`- subpredicates` (page 234)
Returns the array of the receiver's subpredicates.

Class Methods

andPredicateWithSubpredicates:

Returns a new predicate formed by AND-ing the predicates in a given array.

```
+ (NSPredicate *)andPredicateWithSubpredicates:(NSArray *)subpredicates
```

Parameters

subpredicates
An array of `NSPredicate` objects.

Return Value
A new predicate formed by AND-ing the predicates specified by *subpredicates*.

Discussion
An AND predicate with no subpredicates evaluates to TRUE.

Special Considerations

For applications linked on Mac OS X v10.5 or later, the *subpredicates* array is copied. For applications linked on Mac OS X v10.4, the *subpredicates* array is retained (for binary compatibility).

Availability
Available in iOS 3.0 and later.

Declared In
`NSCompoundPredicate.h`

notPredicateWithSubpredicate:

Returns a new predicate formed by NOT-ing a given predicate.

```
+ (NSPredicate *)notPredicateWithSubpredicate:(NSPredicate *)predicate
```

Parameters

predicate

A predicate.

Return Value

A new predicate formed by NOT-ing the predicate specified by *predicate*.

Special Considerations

For applications linked on Mac OS X v10.5 or later, the *subpredicates* array is copied. For applications linked on Mac OS X v10.4, the *subpredicates* array is retained (for binary compatibility).

Availability

Available in iOS 3.0 and later.

Declared In

NSCompoundPredicate.h

orPredicateWithSubpredicates:

Returns a new predicate formed by OR-ing the predicates in a given array.

```
+ (NSPredicate *)orPredicateWithSubpredicates:(NSArray *)subpredicates
```

Parameters

subpredicates

An array of NSPredicate objects.

Return Value

A new predicate formed by OR-ing the predicates specified by *subpredicates*.

Discussion

An OR predicate with no subpredicates evaluates to FALSE.

Special Considerations

For applications linked on Mac OS X v10.5 or later, the *subpredicates* array is copied. For applications linked on Mac OS X v10.4, the *subpredicates* array is retained (for binary compatibility).

Availability

Available in iOS 3.0 and later.

Declared In

NSCompoundPredicate.h

Instance Methods

compoundPredicateType

Returns the predicate type for the receiver.

```
- (NSCompoundPredicateType)compoundPredicateType
```

Return Value
The predicate type for the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSCompoundPredicate.h`

initWithType:subpredicates:

Returns the receiver initialized to a given type using predicates from a given array.

```
- (id)initWithType:(NSCompoundPredicateType)type subpredicates:(NSArray
    *)subpredicates
```

Parameters

type
The type of the new predicate.

subpredicates
An array of `NSPredicate` objects.

Return Value
The receiver initialized with its type set to `type` and subpredicates array to *subpredicates*.

Special Considerations

For applications linked on Mac OS X v10.5 or later, the *subpredicates* array is copied. For applications linked on Mac OS X v10.4, the *subpredicates* array is retained (for binary compatibility).

Availability
Available in iOS 3.0 and later.

Declared In
`NSCompoundPredicate.h`

subpredicates

Returns the array of the receiver's subpredicates.

```
- (NSArray *)subpredicates
```

Return Value
The array of the receiver's subpredicates.

Availability
Available in iOS 3.0 and later.

Declared In
`NSCompoundPredicate.h`

Constants

Compound Predicate Types

These constants describe the possible types of `NSCompoundPredicate`.

```
typedef enum {
    NSNotPredicateType = 0,
    NSAndPredicateType,
    NSOrPredicateType,
} NSCompoundPredicateType;
```

Constants

`NSNotPredicateType`

A logical NOT predicate.

Available in iOS 3.0 and later.

Declared in `NSCompoundPredicate.h`.

`NSAndPredicateType`

A logical AND predicate.

Available in iOS 3.0 and later.

Declared in `NSCompoundPredicate.h`.

`NSOrPredicateType`

A logical OR predicate.

Available in iOS 3.0 and later.

Declared in `NSCompoundPredicate.h`.

Declared In

`NSCompoundPredicate.h`

NSCondition Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

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 240)
  Blocks the current thread until the condition is signaled.
- `waitUntilDate:` (page 241)
  Blocks the current thread until the condition is signaled or the specified time limit is reached.

Signaling Waiting Threads

- `signal` (page 240)
  Signals the condition, waking up one thread waiting on it.
- `broadcast` (page 239)
  Signals the condition, waking up all threads waiting on it.

Accessor Methods

- `setName:` (page 239)
  Assigns a name to the receiver.

- name (page 239)
  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 iOS 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 iOS 2.0 and later.

See Also
- setName: (page 239)

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 iOS 2.0 and later.

See Also
- `name` (page 239)

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 iOS 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 iOS 2.0 and later.

See Also
- `lock` (page 1609) (`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 iOS 2.0 and later.

See Also

`- lock` (page 1609) (`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 iOS 2.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

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 1609)
- `unlock` (page 1610)

Tasks

Initializing an NSConditionLock Object

`- initWithCondition:` (page 245)
Initializes a newly allocated `NSConditionLock` object and sets its condition.

Returning the Condition

- `condition` (page 244)
  Returns the condition associated with the receiver.

Acquiring and Releasing a Lock

- `lockBeforeDate:` (page 245)
  Attempts to acquire a lock before a specified moment in time.
- `lockWhenCondition:` (page 245)
  Attempts to acquire a lock.
- `lockWhenCondition:beforeDate:` (page 246)
  Attempts to acquire a lock before a specified moment in time.
- `tryLock` (page 247)
  Attempts to acquire a lock without regard to the receiver's condition.
- `tryLockWhenCondition:` (page 248)
  Attempts to acquire a lock if the receiver's condition is equal to the specified condition.
- `unlockWithCondition:` (page 248)
  Relinquishes the lock and sets the receiver's condition.

Accessor Methods

- `setName:` (page 247)
  Assigns a name to the receiver.
- `name` (page 246)
  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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- lockWhenCondition:beforeDate:` (page 246)

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 iOS 2.0 and later.

See Also

- `lockWhenCondition:beforeDate:` (page 246)
- `unlockWithCondition:` (page 248)

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 iOS 2.0 and later.

See Also

- `lockBeforeDate:` (page 245)
- `lockWhenCondition:` (page 245)

Declared In

`NSLock.h`

name

Returns the name associated with the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setName:` (page 247)

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 iOS 2.0 and later.

See Also
- `name` (page 246)

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 iOS 2.0 and later.

See Also
- `tryLockWhenCondition:` (page 248)

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 246). This method returns immediately.

Availability
Available in iOS 2.0 and later.

See Also
- tryLock (page 247)

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 iOS 2.0 and later.

See Also
- lockWhenCondition: (page 245)

Declared In
NSLock.h

NSCountedSet Class Reference

Inherits from	NSMutableSet : NSSet : NSObject
Conforms to	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSFastEnumeration (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSSet.h
Companion guide	Collections Programming Topics

Overview

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 1143) 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 250) and `removeObject:` (page 253) methods. You can traverse elements of an `NSCountedSet` object using the enumerator returned by `objectEnumerator` (page 253). The `countForObject:` (page 251) method returns the number of times a given object has been added to this set.

While `NSCountedSet` and `CFBag` are not toll-free bridged, they provide similar functionality. For more information on `CFBag`, consult the *CFBag Reference*.

Tasks

Initializing a Counted Set

- `initWithArray:` (page 251)
  Returns a counted set object initialized with the contents of a given array.
- `initWithSet:` (page 252)
  Returns a counted set object initialized with the contents of a given set.
- `initWithCapacity:` (page 252)
  Returns a counted set object initialized with enough memory to hold a given number of objects.

Adding and Removing Entries

- `addObject:` (page 250)
  Adds a given object to the receiver.
- `removeObject:` (page 253)
  Removes a given object from the receiver.

Examining a Counted Set

- `countForObject:` (page 251)
  Returns the count associated with a given object in the receiver.
- `objectEnumerator` (page 253)
  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 1638) message.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also
- count (page 1143) (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 iOS 2.0 and later.

See Also
initWithArray: (page 1146) (NSSet)
setWithArray: (page 1138) (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 iOS 2.0 and later.

See Also

`initWithCapacity:` (page 793) (`NSMutableSet`)

`setWithCapacity:` (page 791) (`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 iOS 2.0 and later.

See Also

`initWithSet:` (page 1147) (NSSet)

`setWithSet:` (page 1140) (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 1141) method to create a "snapshot," then enumerate the snapshot and modify the original set.

Availability
Available in iOS 2.0 and later.

See Also
`nextObject` (page 424) (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 1636) message. `removeObject:` does nothing if *anObject* is not present in the receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- countForObject:` (page 251)

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 iOS 2.0 and later.
Declared in	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guides	Binary Data Programming Guide Property List Programming Guide
Related sample code	CryptoExercise GKRocket GKTank ScrollViewSuite WiTap

Overview

`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 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 `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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating Data Objects

`+ data` (page 258)
Creates and returns an empty data object.

`+ dataWithBytes:length:` (page 258)
Creates and returns a data object containing a given number of bytes copied from a given buffer.

`+ dataWithBytesNoCopy:length:` (page 259)
Creates and returns a data object that holds *length* bytes from the buffer *bytes*.

`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 259)
Creates and returns a data object that holds a given number of bytes from a given buffer.

`+ dataWithContentsOfFile:` (page 260)
Creates and returns a data object by reading every byte from the file specified by a given path.

`+ dataWithContentsOfFile:options:error:` (page 261)
Creates and returns a data object by reading every byte from the file specified by a given path.

`+ dataWithContentsOfMappedFile:` (page 261)
Creates and returns a data object from the mapped file specified by *path*.

`+ dataWithContentsOfURL:` (page 262)
Returns a data object containing the data from the location specified by a given URL.

`+ dataWithContentsOfURL:options:error:` (page 262)
Creates and returns a data object containing the data from the location specified by *aURL*.

`+ dataWithData:` (page 263)
Creates and returns a data object containing the contents of another data object.

`- initWithBytes:length:` (page 266)
Returns a data object initialized by adding to it a given number of bytes of data copied from a given buffer.

`- initWithBytesNoCopy:length:` (page 266)
Returns a data object initialized by adding to it a given number of bytes of data from a given buffer.

`- initWithBytesNoCopy:length:freeWhenDone:` (page 267)
Initializes a newly allocated data object by adding to it *length* bytes of data from the buffer *bytes*.

- `initWithContentsOfFile:` (page 267)
  Returns a data object initialized by reading into it the data from the file specified by a given path.
- `initWithContentsOfFile:options:error:` (page 268)
  Returns a data object initialized by reading into it the data from the file specified by a given path.
- `initWithContentsOfMappedFile:` (page 268)
  Returns a data object initialized by reading into it the mapped file specified by a given path.
- `initWithContentsOfURL:` (page 269)
  Initializes a newly allocated data object initialized with the data from the location specified by *aURL*.
- `initWithContentsOfURL:options:error:` (page 269)
  Returns a data object initialized with the data from the location specified by a given URL.
- `initWithData:` (page 270)
  Returns a data object initialized with the contents of another data object.

Accessing Data

- `bytes` (page 263)
  Returns a pointer to the receiver's contents.
- `description` (page 264)
  Returns an `NSString` object that contains a hexadecimal representation of the receiver's contents.
- `getBytes:length:` (page 265)
  Copies a number of bytes from the start of the receiver's data into a given buffer.
- `getBytes:range:` (page 265)
  Copies a range of bytes from the receiver's data into a given buffer.
- `subdataWithRange:` (page 272)
  Returns a data object containing a copy of the receiver's bytes that fall within the limits specified by a given range.
- `rangeOfData:options:range:` (page 271)
  Finds and returns the range of the first occurrence of the given data, within the given range, subject to given options.
- `getBytes:` (page 264) Deprecated in iOS 4.0
  Copies a data object's contents into a given buffer. (Deprecated. This method is unsafe because it could potentially cause buffer overruns. You should use `getBytes:length:` (page 265) or `getBytes:range:` (page 265) instead.)

Testing Data

- `isEqualToData:` (page 270)
  Compares the receiving data object to *otherData*.
- `length` (page 271)
  Returns the number of bytes contained in the receiver.

Storing Data

- `writeToFile:atomically:` (page 272)
  Writes the bytes in the receiver to the file specified by a given path.
- `writeToFile:options:error:` (page 273)
  Writes the bytes in the receiver to the file specified by a given path.
- `writeToURL:atomically:` (page 273)
  Writes the bytes in the receiver to the location specified by *aURL*.
- `writeToURL:options:error:` (page 274)
  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 iOS 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 iOS 2.0 and later.

See Also
`+ dataWithBytesNoCopy:length:` (page 259)
`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 259)

Related Sample Code
CryptoExercise
GKTank
WiTap

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 iOS 2.0 and later.

See Also
`+ dataWithBytes:length:` (page 258)
`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 259)

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 iOS 2.0 and later.

See Also

+ dataWithBytes:length: (page 258)

+ dataWithBytesNoCopy:length: (page 259)

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 261) with no options. If you need to know what was the reason for failure, use dataWithContentsOfFile:options:error: (page 261).

A sample using this method can be found in "Working With Binary Data".

Availability

Available in iOS 2.0 and later.

See Also

+ dataWithContentsOfFile:options:error: (page 261)

+ dataWithContentsOfMappedFile: (page 261)

Related Sample Code

ScrollViewSuite

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:(NSDataReadingOptions)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 "NSDataReadingOptions" (page 275).

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 iOS 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 260) 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 iOS 2.0 and later.

See Also
`+ dataWithContentsOfFile:` (page 260)

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 262).

Availability
Available in iOS 2.0 and later.

See Also
`+ dataWithContentsOfURL:options:error:` (page 262)
`- initWithContentsOfURL:` (page 269)

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:(NSDataReadingOptions)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 "`NSDataReadingOptions`" (page 275).

errorPtr

If there is an error reading in the data, upon return contains an `NSError` object that describes the problem.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithContentsOfURL:` (page 269)

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 iOS 2.0 and later.

See Also
- `initWithData:` (page 270)

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.

Discussion
If the `length` (page 271) of the receiver is 0, this method returns `nil`.

Availability
Available in iOS 2.0 and later.

See Also
- `description` (page 264)
- `getBytes:` (page 264)
- `getBytes:length:` (page 265)
- `getBytes:range:` (page 265)

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

See Also
- `bytes` (page 263)
- `getBytes:` (page 264)
- `getBytes:length:` (page 265)
- `getBytes:range:` (page 265)

Declared In
`NSData.h`

getBytes:

Copies a data object's contents into a given buffer. (Deprecated in iOS 4.0. This method is unsafe because it could potentially cause buffer overruns. You should use `getBytes:length:` (page 265) or `getBytes:range:` (page 265) instead.)

```
- (void)getBytes:(void *)buffer
```

Parameters
buffer
A buffer into which to copy the receiver's data. The buffer must be at least `length` (page 271) bytes.

Discussion
You can see a sample using this method in "Working With Binary Data".

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 4.0.

See Also
- `bytes` (page 263)
- `description` (page 264)
- `getBytes:length:` (page 265)
- `getBytes:range:` (page 265)

Declared In
NSData.h

getBytes:length:

Copies a 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*.

Discussion
The number of bytes copied is the smaller of the *length* parameter and the `length` of the data encapsulated in the object.

Availability
Available in iOS 2.0 and later.

See Also
- `bytes` (page 263)
- `description` (page 264)
- `getBytes:` (page 264)
- `getBytes:range:` (page 265)

Related Sample Code
GKRocket

Declared In
NSData.h

getBytes:range:

Copies a range of bytes from the receiver's data into a given buffer.

```
- (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 iOS 2.0 and later.

See Also
- `bytes` (page 263)
- `description` (page 264)
- `getBytes:` (page 264)
- `getBytes:length:` (page 265)

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 iOS 2.0 and later.

See Also
- `+ dataWithBytes:length:` (page 258)
- `- initWithBytesNoCopy:length:` (page 266)
- `- initWithBytesNoCopy:length:freeWhenDone:` (page 267)

Related Sample Code
CryptoExercise
GKRocket

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 iOS 2.0 and later.

See Also
`+ dataWithBytes:length:` (page 258)
`- initWithBytes:length:` (page 266)
`- initWithBytesNoCopy:length:freeWhenDone:` (page 267)

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 iOS 2.0 and later.

See Also
`+ dataWithBytesNoCopy:length:freeWhenDone:` (page 259)
`- initWithBytes:length:` (page 266)
`- initWithBytesNoCopy:length:` (page 266)

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 268) with no options.

Availability
Available in iOS 2.0 and later.

See Also
`+ dataWithContentsOfFile:` (page 260)
`- initWithContentsOfMappedFile:` (page 268)

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:(NSDataReadingOptions)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 "`NSDataReadingOptions`" (page 275).

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 iOS 2.0 and later.

See Also
`+ dataWithContentsOfFile:options:error:` (page 261)

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 iOS 2.0 and later.

See Also

+ dataWithContentsOfMappedFile: (page 261)

- initWithContentsOfFile: (page 267)

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 iOS 2.0 and later.

See Also

+ dataWithContentsOfURL: (page 262)

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:(NSDataReadingOptions)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 "NSDataReadingOptions" (page 275).

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 iOS 2.0 and later.

See Also
+ dataWithContentsOfURL:options:error: (page 262)

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 iOS 2.0 and later.

See Also
+ dataWithData: (page 263)

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 iOS 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 iOS 2.0 and later.

Related Sample Code
CryptoExercise
GKRocket

Declared In
`NSData.h`

rangeOfData:options:range:

Finds and returns the range of the first occurrence of the given data, within the given range, subject to given options.

```
- (NSRange)rangeOfData:(NSData *)dataToFind options:(NSDataSearchOptions)mask
    range:(NSRange)searchRange
```

Parameters

`dataToFind`

The data for which to search. This value must not be `nil`.

Important: Raises an `NSInvalidArgumentException` (page 1773) if *`dataToFind`* is `nil`.

`mask`

A mask specifying search options. The "NSDataSearchOptions" (page 276) options may be specified singly or by combining them with the C bitwise `OR` operator.

searchRange

The range within the receiver in which to search for *dataToFind*. If this range is not within the receiver's range of bytes, an NSRangeException (page 1773) raised.

Return Value
An NSRange (page 1746) structure giving the location and length of *dataToFind* within *searchRange*, modulo the options in *mask*. The range returned is relative to the start of the searched data, not the passed-in search range. Returns {NSNotFound, 0} if *dataToFind* is not found or is empty (@"").

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Related Sample Code
GKRocket

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 1267) 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 iOS 2.0 and later.

See Also
- writeToURL:atomically: (page 273)

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:(NSDataWritingOptions)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 "NSDataWritingOptions" (page 275).

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 iOS 2.0 and later.

See Also
- writeToURL:options:error: (page 274)

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 272), except for the type of the first argument.

Availability
Available in iOS 2.0 and later.

See Also
`- writeToFile:atomically:` (page 272)

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:(NSDataWritingOptions)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 "NSDataWritingOptions" (page 275).

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 273), except for the type of the first argument.

Availability
Available in iOS 2.0 and later.

See Also
`- writeToFile:options:error:` (page 273)

Declared In
NSData.h

Constants

NSDataReadingOptions

Options for methods used to read NSData objects.

```
enum {
    NSDataReadingMapped =   1UL << 0,
    NSDataReadingUncached = 1UL << 1
    NSMappedRead = NSDataReadingMapped,
    NSUncachedRead = NSDataReadingUncached
};
typedef NSUInteger NSDataReadingOptions;
```

Constants

NSDataReadingMapped
: A hint indicating the file should be mapped into virtual memory, if possible.

  Available in iOS 4.0 and later.

  Declared in NSData.h.

NSDataReadingUncached
: 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 iOS 4.0 and later.

  Declared in NSData.h.

NSMappedRead
: Deprecated name for NSDataReadingMapped (page 275). (Deprecated. Please use NSDataReadingMapped (page 275) instead.)

  Available in iOS 2.0 and later.

  Declared in NSData.h.

NSUncachedRead
: Deprecated name for NSDataReadingUncached (page 275). (Deprecated. Please use NSDataReadingUncached (page 275) instead.)

  Available in iOS 2.0 and later.

  Declared in NSData.h.

NSDataWritingOptions

Options for methods used to write NSData objects.

```
enum {
    NSDataWritingAtomic = 1UL << 0
    NSDataWritingFileProtectionNone     = 0x10000000,
    NSDataWritingFileProtectionComplete = 0x20000000,
    NSDataWritingFileProtectionMask     = 0xf0000000,

    NSAtomicWrite = NSDataWritingAtomic
};
typedef NSUInteger NSDataWritingOptions;
```

Constants

`NSDataWritingAtomic`

A hint to write data to an auxiliary file first and then exchange the files. This option is equivalent to using a write method taking the parameter `atomically:YES`.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

`NSDataWritingFileProtectionNone`

A hint to set the content protection attribute of the file when writing it out. In this case, the file is not stored in an encrypted format and may be accessed at boot time and while the device is unlocked.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

`NSDataWritingFileProtectionComplete`

A hint to set the content protection attribute of the file when writing it out. In this case, the file is stored in an encrypted format and may be read from or written to only while the device is unlocked. At all other times, attempts to read and write the file result in failure.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

`NSDataWritingFileProtectionMask`

A mask to use when determining the file protection options assigned to the data.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

`NSAtomicWrite`

Deprecated constant. (Deprecated. Use `NSDataWritingAtomic` (page 276) instead.)

Available in iOS 2.0 and later.

Declared in `NSData.h`.

NSDataSearchOptions

Options for method used to search `NSData` objects. These options are used with the `rangeOfData:options:range:` (page 271) method.

```
enum {
    NSDataSearchBackwards = 1UL << 0,
    NSDataSearchAnchored = 1UL << 1
};
typedef NSUInteger NSDataSearchOptions;
```

Constants

`NSDataSearchBackwards`

Search from the end of `NSData` object.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

`NSDataSearchAnchored`

Search is limited to start (or end, if `NSDataSearchBackwards`) of `NSData` object.

This option performs searching only on bytes at the beginning or end of the range. No match at the beginning or end means nothing is found, even if a matching sequence of bytes occurs elsewhere in the data object.

Available in iOS 4.0 and later.

Declared in `NSData.h`.

NSDataDetector Class Reference

Inherits from	NSRegularExpression : NSObject
Conforms to	NSCoding (NSRegularExpression) NSCopying (NSRegularExpression) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSRegularExpression.h

Overview

The `NSDataDetector` class is a specialized subclass of the `NSRegularExpression` class designed to match data detectors.

Currently the `NSDataDetector` class can match dates, addresses, links, phone numbers and transit information.

The results of matching content is returned as `NSTextCheckingResult` objects. However, the `NSTextCheckingResult` objects returned by `NSDataDetector` are different from those returned by the base class `NSRegularExpression`. Results returned by `NSDataDetector` will be of one of the data detectors types, depending on the type of result being returned, and they will have corresponding properties. For example, results of type `NSTextCheckingTypeDate` (page 1306) have a `date` (page 1292), `timeZone`, and `duration`; results of type `NSTextCheckingTypeLink` (page 1307) have a `URL` (page 1296), and so forth.

Examples

The following shows several graduated examples of using the `NSDataDetector` class.

This code fragment creates a data detector that will find URL links and phone numbers. If an error was encountered it is returned in `error`.

```
   NSError *error = NULL;
   NSDataDetector *detector = [NSDataDetector
dataDetectorWithTypes:NSTextCheckingTypeLink|NSTextCheckingTypePhoneNumber
                                                                  error:&error];
```

Once the data detector instance is created you can determine the number of matches within a range of a string using the `NSRegularExpression` method `numberOfMatchesInString:options:range:` (page 1095).

```
    NSUInteger numberOfMatches = [detector numberOfMatchesInString:string
                                                           options:0
                                                            range:NSMakeRange(0,
 [string length])];
```

If you are interested only in the overall range of the first match, the `numberOfMatchesInString:options:range:` (page 1095) method provides it. However, with data detectors, this is less likely than with regular expressions, because clients usually will be interested in additional information as well.

The additional information available depends on the type of the result. For results of type `NSTextCheckingTypeLink` (page 1307), it is the `URL` property that is significant. For results of type `NSTextCheckingTypePhoneNumber` (page 1307), it is the `phoneNumber` (page 1294) property instead.

The `matchesInString:options:range:` (page 1095) method is similar to the `firstMatchInString:options:range:` (page 1093), except that it returns all matches rather than only the first. The following code fragment finds all the matches for links and phone numbers in a string.

```
    NSArray *matches = [detector matchesInString:string
                                         options:0
                                        range:NSMakeRange(0, [string length])];
    for (NSTextCheckingResult *match in matches) {
         NSRange matchRange = [match range];
         if ([match resultType] == NSTextCheckingTypeLink) {
             NSURL *url = [match URL];
         } else if ([match resultType] == NSTextCheckingTypePhoneNumber) {
             NSString *phoneNumber = [match phoneNumber];
         }
    }
```

The `NSRegularExpression` Block object enumerator is the most general and flexible of the matching methods. It allows you to iterate through matches in a string, performing arbitrary actions on each as specified by the code in the Block, and to stop partway through if desired. In the following code fragment, the iteration is stopped after a certain number of matches have been found.

```
    __block NSUInteger count = 0;
    [detector enumerateMatchesInString:string
                               options:0
                                 range:NSMakeRange(0, [string length])
                            usingBlock:^(NSTextCheckingResult *match,
NSMatchingFlags flags, BOOL *stop){
         NSRange matchRange = [match range];
         if ([match resultType] == NSTextCheckingTypeLink) {
             NSURL *url = [match URL];
         } else if ([match resultType] == NSTextCheckingTypePhoneNumber) {
             NSString *phoneNumber = [match phoneNumber];
         }
         if (++count >= 100) *stop = YES;
    }];
```

Tasks

Creating Data Detector Instances

`+ dataDetectorWithTypes:error:` (page 282)
Creates and returns a new data detector instance.

`- initWithTypes:error:` (page 282)
Initializes and returns a data detector instance.

Getting the Checking Types

`checkingTypes` (page 281) *property*
Returns the checking types for this data detector. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

checkingTypes

Returns the checking types for this data detector. (read-only)

```
@property(readonly) NSTextCheckingTypes checkingTypes
```

Discussion
The supported subset of checking types are specified in `NSTextCheckingType` (page 1306). Those constants can be combined using the C-bitwise OR operator.

Currently, the supported data detectors *`checkingTypes`* are: `NSTextCheckingTypeDate` (page 1306), `NSTextCheckingTypeAddress` (page 1306), `NSTextCheckingTypeLink` (page 1307), `NSTextCheckingTypePhoneNumber` (page 1307), and `NSTextCheckingTypeTransitInformation` (page 1307).

Availability
Available in iOS 4.0 and later.

Declared In
`NSRegularExpression.h`

Class Methods

dataDetectorWithTypes:error:

Creates and returns a new data detector instance.

```
+ (NSDataDetector *)dataDetectorWithTypes:(NSTextCheckingTypes)checkingTypes
    error:(NSError **)error
```

Parameters

checkingTypes

The checking types. The supported checking types are a subset of the types specified in `NSTextCheckingType` (page 1306). Those constants can be combined using the C-bitwise OR operator.

error

An out parameter that if an error occurs during initialization contains the encountered error.

Return Value

Returns the newly initialized data detector. If an error was encountered returns `nil`, and *error* contains the error.

Discussion

Currently, the supported data detectors *checkingTypes* are: `NSTextCheckingTypeDate` (page 1306), `NSTextCheckingTypeAddress` (page 1306), `NSTextCheckingTypeLink` (page 1307), `NSTextCheckingTypePhoneNumber` (page 1307), and `NSTextCheckingTypeTransitInformation` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`- initWithTypes:error:` (page 282)
`@property checkingTypes` (page 281)

Declared In

`NSRegularExpression.h`

Instance Methods

initWithTypes:error:

Initializes and returns a data detector instance.

```
- (id)initWithTypes:(NSTextCheckingTypes)checkingTypes error:(NSError **)error
```

Parameters

checkingTypes

The checking types. The supported checking types are a subset of the types `NSTextCheckingType` (page 1306). Those constants can be combined using the C-bitwise OR operator.

error

An out parameter that if an error occurs during initialization contains the encountered error.

Return Value

Returns the newly initialized data detector. If an error was encountered returns `nil`, and *error* contains the error.

Discussion

Currently, the supported data detectors *checkingTypes* are: `NSTextCheckingTypeDate` (page 1306), `NSTextCheckingTypeAddress` (page 1306), `NSTextCheckingTypeLink` (page 1307), `NSTextCheckingTypePhoneNumber` (page 1307), and `NSTextCheckingTypeTransitInformation` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`+ dataDetectorWithTypes:error:` (page 282)

`@property checkingTypes` (page 281)

Declared In

`NSRegularExpression.h`

NSDate Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDate.h Foundation/NSCalendarDate.h
Companion guides	Date and Time Programming Guide Property List Programming Guide
Related sample code	aurioTouch GKRocket GKTank

Overview

`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 300), 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.

To parse strings containing dates and to generate string representations of a date, you should use an instance of `NSDateFormatter` using the methods `dateFromString:` (page 329) and `stringFromDate:` (page 353) respectively—see Date Formatters 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 300) instance method to provide the correct date and time value based on your instance variable.
- Override `initWithTimeIntervalSinceReferenceDate:` (page 297), 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 300) and `initWithTimeIntervalSinceReferenceDate:` (page 297). 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 1552)
`initWithCoder:` (page 1552)

NSCopying
`copyWithZone:` (page 1554)

Tasks

Creating and Initializing Date Objects

`+ date` (page 289)
Creates and returns a new date set to the current date and time.

`+ dateWithTimeIntervalSinceNow:` (page 290)
Creates and returns an `NSDate` object set to a given number of seconds from the current date and time.

`+ dateWithTimeInterval:sinceDate:` (page 289)
Creates and returns an `NSDate` object set to a given number of seconds from the specified date.

`+ dateWithTimeIntervalSinceReferenceDate:` (page 290)
Creates and returns an `NSDate` object set to a given number of seconds from the first instant of 1 January 2001, GMT.

`+ dateWithTimeIntervalSince1970:` (page 290)
Creates and returns an `NSDate` object set to the given number of seconds from the first instant of 1 January 1970, GMT.

`- init` (page 296)
Returns an `NSDate` object initialized to the current date and time.

`- initWithTimeIntervalSinceNow:` (page 297)
Returns an `NSDate` object initialized relative to the current date and time by a given number of seconds.

`- initWithTimeInterval:sinceDate:` (page 296)
Returns an `NSDate` object initialized relative to another given date by a given number of seconds.

`- initWithTimeIntervalSinceReferenceDate:` (page 297)
Returns an `NSDate` object initialized relative the first instant of 1 January 2001, GMT by a given number of seconds.

`- initWithTimeIntervalSince1970:` (page 296)
Returns an `NSDate` object set to the given number of seconds from the first instant of 1 January 1970, GMT.

Getting Temporal Boundaries

`+ distantFuture` (page 291)
Creates and returns an `NSDate` object representing a date in the distant future.

`+ distantPast` (page 291)
Creates and returns an `NSDate` object representing a date in the distant past.

Comparing Dates

- `isEqualToDate:` (page 298)
  Returns a Boolean value that indicates whether a given object is an `NSDate` object and exactly equal the receiver.
- `earlierDate:` (page 295)
  Returns the earlier of the receiver and another given date.
- `laterDate:` (page 298)
  Returns the later of the receiver and another given date.
- `compare:` (page 293)
  Returns an `NSComparisonResult` value that indicates the temporal ordering of the receiver and another given date.

Getting Time Intervals

- `timeIntervalSinceDate:` (page 299)
  Returns the interval between the receiver and another given date.
- `timeIntervalSinceNow` (page 300)
  Returns the interval between the receiver and the current date and time.
+ `timeIntervalSinceReferenceDate` (page 292)
  Returns the interval between the first instant of 1 January 2001, GMT and the current date and time.
- `timeIntervalSinceReferenceDate` (page 300)
  Returns the interval between the receiver and the first instant of 1 January 2001, GMT.
- `timeIntervalSince1970` (page 299)
  Returns the interval between the receiver and the first instant of 1 January 1970, GMT.

Adding a Time Interval

- `dateByAddingTimeInterval:` (page 294)
  Returns a new `NSDate` object that is set to a given number of seconds relative to the receiver.
- `addTimeInterval:` (page 292) Deprecated in iOS 4.0
  Returns a new `NSDate` object that is set to a given number of seconds relative to the receiver. (Deprecated. This method has been replaced by `dateByAddingTimeInterval:` (page 294).)

Representing Dates as Strings

- `description` (page 294)
  Returns a string representation of the receiver.
- `descriptionWithLocale:` (page 294)
  Returns a string representation of the receiver using the given locale.

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 296).

The following code sample shows how to use `date` to get the current date:

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

Availability
Available in iOS 2.0 and later.

Related Sample Code
GKTank

Declared In
`NSDate.h`

dateWithTimeInterval:sinceDate:

Creates and returns an `NSDate` object set to a given number of seconds from the specified date.

```
+ (id)dateWithTimeInterval:(NSTimeInterval)seconds sinceDate:(NSDate *)date
```

Parameters

seconds
The number of seconds to add to *date*. Use a negative argument to specify a date and time before *date*.

date
The date.

Return Value
An `NSDate` object set to *seconds* seconds from *date*.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also

`- timeIntervalSince1970` (page 299)

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 iOS 2.0 and later.

See Also

`- initWithTimeIntervalSinceNow:` (page 297)

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 iOS 2.0 and later.

See Also
`- initWithTimeIntervalSinceReferenceDate:` (page 297)

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 iOS 2.0 and later.

See Also
`+ distantPast` (page 291)

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 iOS 2.0 and later.

See Also
`+ distantFuture` (page 291)

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 iOS 2.0 and later.

See Also
`- timeIntervalSinceReferenceDate` (page 300)
`- timeIntervalSinceDate:` (page 299)
`- timeIntervalSince1970` (page 299)
`- timeIntervalSinceNow` (page 300)

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. (Deprecated in iOS 4.0. This method has been replaced by `dateByAddingTimeInterval:` (page 294).)

```
- (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 iOS 2.0 and later.

Deprecated in iOS 4.0.

See Also

- `initWithTimeInterval:sinceDate:` (page 296)
- `timeIntervalSinceDate:` (page 299)
- `dateByAddingTimeInterval:` (page 294)

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 299) to compare the two dates.

Availability

Available in iOS 2.0 and later.

See Also

- `earlierDate:` (page 295)
- `isEqual:` (page 1632) (`NSObject` protocol)
- `laterDate:` (page 298)

Declared In
NSDate.h

dateByAddingTimeInterval:

Returns a new NSDate object that is set to a given number of seconds relative to the receiver.

```
- (id)dateByAddingTimeInterval:(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 iOS 4.0 and later.

See Also
- initWithTimeInterval:sinceDate: (page 296)
- timeIntervalSinceDate: (page 299)

Declared In
NSDate.h

description

Returns a string representation of the receiver.

```
- (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").

Availability
Available in iOS 2.0 and later.

See Also
- descriptionWithLocale: (page 294)

Declared In
NSDate.h

descriptionWithLocale:

Returns a string representation of the receiver using the given locale.

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

Parameters

locale

An `NSLocale` object.

If you pass `nil`, `NSDate` formats the date in the same way as the `description` (page 294) method.

On Mac OS X v10.4 and earlier, this parameter was an `NSDictionary` object. If you pass in an `NSDictionary` object on Mac OS X v10.5, `NSDate` uses the default user locale—the same as if you passed in `[NSLocale currentLocale]`.

Return Value

A string representation of the receiver, using the given locale, or if the locale argument is `nil`, 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

On Mac OS X v10.4 and earlier, *localeDictionary* is an `NSDictionary` object containing locale data. To use the user's preferences, you can use `[[NSUserDefaults standardUserDefaults] dictionaryRepresentation]`.

Availability

Available in iOS 4.0 and later.

See Also

- `description` (page 294)

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 299). If the receiver and *anotherDate* represent the same date, returns the receiver.

Availability

Available in iOS 2.0 and later.

See Also

- `compare:` (page 293)
- `isEqual:` (page 1632) (`NSObject` protocol)
- `laterDate:` (page 298)

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 297).

Availability
Available in iOS 2.0 and later.

See Also
`+ date` (page 289)
`- initWithTimeIntervalSinceReferenceDate:` (page 297)

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 297).

Availability
Available in iOS 2.0 and later.

Declared In
`NSDate.h`

initWithTimeIntervalSince1970:

Returns an `NSDate` object set to the given number of seconds from the first instant of 1 January 1970, GMT.

```
- (id)initWithTimeIntervalSince1970:(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 iOS 4.0 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 297).

Availability

Available in iOS 2.0 and later.

See Also

`+ dateWithTimeIntervalSinceNow:` (page 290)

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 iOS 2.0 and later.

See Also
`+ dateWithTimeIntervalSinceReferenceDate:` (page 290)

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 299) to compare the two dates.

Availability
Available in iOS 2.0 and later.

See Also
- `compare:` (page 293)
- `earlierDate:` (page 295)
- `isEqual:` (page 1632) (`NSObject` protocol)
- `laterDate:` (page 298)

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 299). If the receiver and *anotherDate* represent the same date, returns the receiver.

Availability

Available in iOS 2.0 and later.

See Also

- `compare:` (page 293)
- `earlierDate:` (page 295)
- `isEqual:` (page 1632) (`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 iOS 2.0 and later.

See Also

- `timeIntervalSinceDate:` (page 299)
- `timeIntervalSinceNow` (page 300)
- `timeIntervalSinceReferenceDate` (page 300)
+ `timeIntervalSinceReferenceDate` (page 292)

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 iOS 2.0 and later.

See Also
- `timeIntervalSince1970` (page 299)
- `timeIntervalSinceNow` (page 300)
- `timeIntervalSinceReferenceDate` (page 300)

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 iOS 2.0 and later.

See Also
- `timeIntervalSinceDate:` (page 299)
- `timeIntervalSince1970` (page 299)
- `timeIntervalSinceReferenceDate` (page 300)

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 iOS 2.0 and later.

See Also
- `timeIntervalSinceDate:` (page 299)
- `timeIntervalSinceNow` (page 300)

`+ timeIntervalSinceReferenceDate` (page 292)

Related Sample Code
aurioTouch
GKRocket

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 iOS 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`

Notifications

NSSystemClockDidChangeNotification

Posted whenever the system clock is changed. This can be initiated by a call to `settimeofday()` or the user changing values in the Date and Time Preference panel. The notification object is `null`. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 4.0 and later.

Declared In
`NSDate.h`

NSDateComponents Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSCalendar.h
Companion guide	Date and Time Programming Guide

Overview

`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*.

Tasks

Getting Information About an NSDateComponents Object

- `era` (page 306)
  Returns the number of era units for the receiver.
- `year` (page 316)
  Returns the number of year units for the receiver.
- `month` (page 308)
  Returns the number of month units for the receiver.
- `date` (page 306)
  Returns the date of the receiver.
- `day` (page 306)
  Returns the number of day units for the receiver.
- `hour` (page 307)
  Returns the number of hour units for the receiver.
- `minute` (page 307)
  Returns the number of minute units for the receiver.
- `second` (page 308)
  Returns the number of second units for the receiver.
- `week` (page 315)
  Returns the number of week units for the receiver.
- `weekday` (page 315)
  Returns the number of weekday units for the receiver.
- `weekdayOrdinal` (page 316)
  Returns the ordinal number of weekday units for the receiver.
- `calendar` (page 305)
  Returns the calendar of the receiver.
- `timeZone` (page 314)
  Returns the receiver's time zone.

Setting Information for an NSDateComponents Object

- `setEra:` (page 310)
  Sets the number of era units for the receiver.
- `setYear:` (page 314)
  Sets the number of year units for the receiver.

- `setMonth:` (page 311)
  Sets the number of month units for the receiver.
- `setDay:` (page 309)
  Sets the number of day units for the receiver.
- `setHour:` (page 310)
  Sets the number of hour units for the receiver.
- `setMinute:` (page 310)
  Sets the number of minute units for the receiver.
- `setSecond:` (page 312)
  Sets the number of second units for the receiver.
- `setWeek:` (page 313)
  Sets the number of week units for the receiver.
- `setWeekday:` (page 313)
  Sets the number of weekday units for the receiver.
- `setWeekdayOrdinal:` (page 313)
  Sets the ordinal number of weekday units for the receiver.
- `quarter` (page 308)
  Returns the number of quarters in the calendar.
- `setQuarter:` (page 311)
  Sets the number of quarters in the calendar.
- `setCalendar:` (page 309)
  Sets the receiver's calendar.
- `setTimeZone:` (page 312)
  Sets the receiver's time zone.

Instance Methods

calendar

Returns the calendar of the receiver.

```
- (NSCalendar *)calendar
```

Return Value
The calendar.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`- setCalendar:` (page 309)

Declared In
`NSCalendar.h`

date

Returns the date of the receiver.

```
- (NSDate *)date
```

Return Value
The date.

Availability
Available in iOS 4.0 and later.

Declared In
NSCalendar.h

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*.

Availability
Available in iOS 2.0 and later.

See Also
- setDay: (page 309)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- setEra: (page 310)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- setHour: (page 310)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- setMinute: (page 310)

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*.

Availability
Available in iOS 2.0 and later.

See Also
`- setMonth:` (page 311)

Declared In
`NSCalendar.h`

quarter

Returns the number of quarters in the calendar.

```
- (NSInteger)quarter
```

Return Value
The number of quarters units for the receiver.

Availability
Available in iOS 4.0 and later.

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*.

Availability
Available in iOS 2.0 and later.

See Also
- `setSecond:` (page 312)

Declared In
`NSCalendar.h`

setCalendar:

Sets the receiver's calendar.

```
- (void)setCalendar:(NSCalendar *)cal
```

Parameters

cal

The calendar.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability
Available in iOS 4.0 and later.

See Also
- `calendar` (page 305)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- `day` (page 306)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- era (page 306)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- hour (page 307)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- `minute` (page 307)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- `month` (page 308)

Declared In
`NSCalendar.h`

setQuarter:

Sets the number of quarters in the calendar.

```
- (void)setQuarter:(NSInteger)v
```

Parameters

v

The number of quarters 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*.

Availability
Available in iOS 4.0 and later.

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*.

Availability
Available in iOS 2.0 and later.

See Also
- second (page 308)

Declared In
NSCalendar.h

setTimeZone:

Sets the receiver's time zone.

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

Parameters

tz
The time zone.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability
Available in iOS 4.0 and later.

See Also
- timeZone (page 314)

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*.

Availability

Available in iOS 2.0 and later.

See Also

- week (page 315)

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

Weekday units are the numbers 1 through *n*, where *n* is the number of days in the week. For example, in the Gregorian calendar, *n* is 7 and Sunday is represented by 1.

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability

Available in iOS 2.0 and later.

See Also

- weekday (page 315)

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

Weekday ordinal units represent the position of the weekday within the next larger calendar unit, such as the month. For example, *2* is the weekday ordinal unit for the *second* Friday of the month.

This value will be interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability

Available in iOS 2.0 and later.

See Also

`- weekdayOrdinal` (page 316)

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*.

Availability

Available in iOS 2.0 and later.

See Also

`- year` (page 316)

Declared In

`NSCalendar.h`

timeZone

Returns the receiver's time zone.

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

Return Value

The time zone.

Discussion
This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability
Available in iOS 4.0 and later.

See Also
`- setTimeZone:` (page 312)

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*.

Availability
Available in iOS 2.0 and later.

See Also
`- setWeek:` (page 313)

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
Weekday units are the numbers 1 through *n*, where *n* is the number of days in the week. For example, in the Gregorian calendar, *n* is 7 and Sunday is represented by 1.

This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
- `setWeekday:` (page 313)

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
Weekday ordinal units represent the position of the weekday within the next larger calendar unit, such as the month. For example, *2* is the weekday ordinal unit for the *second* Friday of the month.

This value is interpreted in the context of the calendar with which it is used—see Calendars in *Date and Time Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
- `setWeekdayOrdinal:` (page 313)

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*.

Availability
Available in iOS 2.0 and later.

See Also
- `setYear:` (page 314)

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 iOS 2.0 and later.

Declared in `NSCalendar.h`.

Declared In

`NSCalendar.h`

NSDateFormatter Class Reference

Inherits from	NSFormatter : NSObject
Conforms to	NSCoding (NSFormatter) NSCopying (NSFormatter) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDateFormatter.h
Companion guide	Data Formatting Guide

Overview

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 using pre-set format styles or custom format strings.

In general, you are encouraged to use format styles (see `timeStyle` (page 354), `dateStyle` (page 329), and `NSDateFormatterStyle` (page 357)) rather than using custom format strings, since the format for a given style reflects a user's preferences. Format styles also reflect the locale setting.

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];
[dateFormatter setDateStyle:NSDateFormatterMediumStyle];

NSDate *date = [NSDate dateWithTimeIntervalSinceReferenceDate:118800];

NSLocale *usLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US"];
[dateFormatter setLocale:usLocale];

NSLog(@"Date for locale %@: %@",
      [[dateFormatter locale] localeIdentifier], [dateFormatter
stringFromDate:date]);
// Output:
// Date for locale en_US: Jan 2, 2001

NSLocale *frLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"fr_FR"];
[dateFormatter setLocale:frLocale];
NSLog(@"Date for locale %@: %@",
      [[dateFormatter locale] localeIdentifier], [dateFormatter
stringFromDate:date]);
// Output:
```

```
// Date for locale fr_FR: 2 janv. 2001
```

Formatter Behaviors and OS Versions

With Mac OS X v10.4 and later, `NSDateFormatter` has two modes of operation (or behaviors). See *Data Formatting Guide* for a full description of the old and new behaviors.

iOS Note: iOS supports only the 10.4+ behavior. 10.0-style methods and format strings are not available on iOS.

By default, on Mac OS X v10.4 instances of `NSDateFormatter` have the same behavior as they did on Mac OS X versions 10.0 to 10.3. On Mac OS X v10.5 and later, `NSDateFormatter` defaults to the 10.4+ behavior.

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 332). If necessary, you can set the default class behavior using `setDefaultFormatterBehavior:` (page 327)), you can set the behavior for an instance using `setFormatterBehavior:` (page 338) message with the argument `NSDateFormatterBehavior10_4`.

By default, the 10.4-style formatter returns `NSDate` objects (prior to Mac OS X v10.4, date formatters returned `NSCalendarDate` objects). You can change this behavior using `setGeneratesCalendarDates:` (page 339), although this is strongly discouraged (as `NSCalendarDate` is deprecated on Mac OS X v10.6 and later).

Tasks

Initializing a Date Formatter

- `init` (page 332) Available in iOS 2.0 through iOS 3.2
  Initializes and returns an `NSDateFormatter` instance.

Managing Behavior

- `formatterBehavior` (page 331)
  Returns the formatter behavior for the receiver.
- `setFormatterBehavior:` (page 338)
  Sets the formatter behavior for the receiver.
+ `defaultFormatterBehavior` (page 326)
  Returns the default formatting behavior for instances of the class.
+ `setDefaultFormatterBehavior:` (page 327)
  Sets the default formatting behavior for instances of the class.
- `generatesCalendarDates` (page 331)
  Returns a Boolean value that indicates whether the receiver generates calendar dates.

- `setGeneratesCalendarDates:` (page 339)
  Sets whether the receiver generates calendar dates.
- `isLenient` (page 333)
  Returns a Boolean value that indicates whether the receiver uses heuristics when parsing a string.
- `setLenient:` (page 339)
  Sets whether the receiver uses heuristics when parsing a string.
- `doesRelativeDateFormatting` (page 330)
  Returns a Boolean value that indicates whether the receiver uses phrases such as “today” and “tomorrow” for the date component.
- `setDoesRelativeDateFormatting:` (page 337)
  Specifies whether the receiver uses phrases such as “today” and “tomorrow” for the date component.

Converting Objects

- `dateFromString:` (page 329)
  Returns a date representation of a given string interpreted using the receiver’s current settings.
- `stringFromDate:` (page 353)
  Returns a string representation of a given date formatted using the receiver’s current settings.
+ `localizedStringFromDate:dateStyle:timeStyle:` (page 326)
  Returns string representation of a given date formatted for the current locale using the specified date and time styles.
- `getObjectValue:forString:range:error:` (page 331)
  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 328)
  Returns the date format string used by the receiver.
- `setDateFormat:` (page 336)
  Sets the date format for the receiver.
- `dateStyle` (page 329)
  Returns the date style of the receiver.
- `setDateStyle:` (page 336)
  Sets the date style of the receiver.
- `timeStyle` (page 354)
  Returns the time style of the receiver.
- `setTimeStyle:` (page 346)
  Sets the time style of the receiver.
+ `dateFormatFromTemplate:options:locale:` (page 325)
  Returns a localized date format string representing the given date format components arranged appropriately for the specified locale.

Managing Attributes

- `calendar` (page 328)
  Returns the calendar for the receiver.
- `setCalendar:` (page 336)
  Sets the calendar for the receiver.
- `defaultDate` (page 329)
  Returns the default date for the receiver.
- `setDefaultDate:` (page 337)
  Sets the default date for the receiver.
- `locale` (page 333)
  Returns the locale for the receiver.
- `setLocale:` (page 340)
  Sets the locale for the receiver.
- `timeZone` (page 354)
  Returns the time zone for the receiver.
- `setTimeZone:` (page 346)
  Sets the time zone for the receiver.
- `twoDigitStartDate` (page 354)
  Returns the earliest date that can be denoted by a two-digit year specifier.
- `setTwoDigitStartDate:` (page 347)
  Sets the two-digit start date for the receiver.
- `gregorianStartDate` (page 332)
  Returns the start date of the Gregorian calendar for the receiver.
- `setGregorianStartDate:` (page 339)
  Sets the start date of the Gregorian calendar for the receiver.

Managing AM and PM Symbols

- `AMSymbol` (page 327)
  Returns the AM symbol for the receiver.
- `setAMSymbol:` (page 335)
  Sets the AM symbol for the receiver.
- `PMSymbol` (page 334)
  Returns the PM symbol for the receiver.
- `setPMSymbol:` (page 341)
  Sets the PM symbol for the receiver.

Managing Weekday Symbols

- `weekdaySymbols` (page 357)
  Returns the array of weekday symbols for the receiver.

- `setWeekdaySymbols:` (page 349)
  Sets the weekday symbols for the receiver.
- `shortWeekdaySymbols` (page 352)
  Returns the array of short weekday symbols for the receiver.
- `setShortWeekdaySymbols:` (page 344)
  Sets the short weekday symbols for the receiver.
- `veryShortWeekdaySymbols` (page 356)
  Returns the array of very short weekday symbols for the receiver.
- `setVeryShortWeekdaySymbols:` (page 348)
  Sets the vert short weekday symbols for the receiver
- `standaloneWeekdaySymbols` (page 353)
  Returns the array of standalone weekday symbols for the receiver.
- `setStandaloneWeekdaySymbols:` (page 345)
  Sets the standalone weekday symbols for the receiver.
- `shortStandaloneWeekdaySymbols` (page 351)
  Returns the array of short standalone weekday symbols for the receiver.
- `setShortStandaloneWeekdaySymbols:` (page 344)
  Sets the short standalone weekday symbols for the receiver.
- `veryShortStandaloneWeekdaySymbols` (page 356)
  Returns the array of very short standalone weekday symbols for the receiver.
- `setVeryShortStandaloneWeekdaySymbols:` (page 348)
  Sets the very short standalone weekday symbols for the receiver.

Managing Month Symbols

- `monthSymbols` (page 334)
  Returns the month symbols for the receiver.
- `setMonthSymbols:` (page 341)
  Sets the month symbols for the receiver.
- `shortMonthSymbols` (page 349)
  Returns the array of short month symbols for the receiver.
- `setShortMonthSymbols:` (page 342)
  Sets the short month symbols for the receiver.
- `veryShortMonthSymbols` (page 355)
  Returns the very short month symbols for the receiver.
- `setVeryShortMonthSymbols:` (page 347)
  Sets the very short month symbols for the receiver.
- `standaloneMonthSymbols` (page 352)
  Returns the standalone month symbols for the receiver.
- `setStandaloneMonthSymbols:` (page 345)
  Sets the standalone month symbols for the receiver.
- `shortStandaloneMonthSymbols` (page 350)
  Returns the short standalone month symbols for the receiver.

- `setShortStandaloneMonthSymbols:` (page 343)
  Sets the short standalone month symbols for the receiver.
- `veryShortStandaloneMonthSymbols` (page 355)
  Returns the very short month symbols for the receiver.
- `setVeryShortStandaloneMonthSymbols:` (page 347)
  Sets the very short standalone month symbols for the receiver.

Managing Quarter Symbols

- `quarterSymbols` (page 335)
  Returns the quarter symbols for the receiver.
- `setQuarterSymbols:` (page 341)
  Sets the quarter symbols for the receiver.
- `shortQuarterSymbols` (page 350)
  Returns the short quarter symbols for the receiver.
- `setShortQuarterSymbols:` (page 342)
  Sets the short quarter symbols for the receiver.
- `standaloneQuarterSymbols` (page 352)
  Returns the standalone quarter symbols for the receiver.
- `setStandaloneQuarterSymbols:` (page 345)
  Sets the standalone quarter symbols for the receiver.
- `shortStandaloneQuarterSymbols` (page 351)
  Returns the short standalone quarter symbols for the receiver.
- `setShortStandaloneQuarterSymbols:` (page 343)
  Sets the short standalone quarter symbols for the receiver.

Managing Era Symbols

- `eraSymbols` (page 330)
  Returns the era symbols for the receiver.
- `setEraSymbols:` (page 338)
  Sets the era symbols for the receiver.
- `longEraSymbols` (page 334)
  Returns the long era symbols for the receiver
- `setLongEraSymbols:` (page 340)
  Sets the long era symbols for the receiver.

Class Methods

dateFormatFromTemplate:options:locale:

Returns a localized date format string representing the given date format components arranged appropriately for the specified locale.

```
+ (NSString *)dateFormatFromTemplate:(NSString
    *)templateoptions:(NSUInteger)optslocale:(NSLocale *)locale
```

Parameters

template

A string containing date format patterns (such as “MM” or “h”).

For full details, see Unicode Technical Standard #35.

opts

No options are currently defined—pass `0`.

locale

The locale for which the template is required.

Return Value

A localized date format string representing the date format components given in *template*, arranged appropriately for the locale specified by *locale*.

The returned string may not contain exactly those components given in *template*, but may—for example—have locale-specific adjustments applied.

Discussion

Different locales have different conventions for the ordering of date components. You use this method to get an appropriate format string for a given set of components for a specified locale (typically you use the current locale—see `currentLocale` (page 692)).

The following example shows the difference between the date formats for British and American English:

```
NSLocale *usLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US"];
NSLocale *gbLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_GB"];

NSString *dateFormat;
NSString *dateComponents = @"yMMMMd";

dateFormat = [NSDateFormatter dateFormatFromTemplate:dateComponents options:0
locale:usLocale];
NSLog(@"Date format for %@: %@",
    [usLocale displayNameForKey:NSLocaleIdentifier value:[usLocale
localeIdentifier]], dateFormat);

dateFormat = [NSDateFormatter dateFormatFromTemplate:dateComponents options:0
locale:gbLocale];
NSLog(@"Date format for %@: %@",
    [gbLocale displayNameForKey:NSLocaleIdentifier value:[gbLocale
localeIdentifier]], dateFormat);

// Output:
// Date format for English (United States): MMMM d, y
```

```
// Date format for English (United Kingdom): d MMMM y
```

Availability
Available in iOS 4.0 and later.

Declared In
NSDateFormatter.h

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 358).

Discussion
The default is NSDateFormatterBehavior10_0.

Availability
Available in iOS 2.0 and later.

See Also
+ setDefaultFormatterBehavior: (page 327).
- formatterBehavior (page 331)
- setFormatterBehavior: (page 338)

Declared In
NSDateFormatter.h

localizedStringFromDate:dateStyle:timeStyle:

Returns string representation of a given date formatted for the current locale using the specified date and time styles.

```
+ (NSString *)localizedStringFromDate:(NSDate
    *)date dateStyle:(NSDateFormatterStyle)dateStyle timeStyle:(NSDateFormatterStyle)timeStyle
```

Parameters

date
A date.

dateStyle
A format style for the date. For possible values, see NSDateFormatterStyle (page 357).

timeStyle
A format style for the time. For possible values, see NSDateFormatterStyle (page 357).

Return Value
A localized string representation of *date* using the specified date and time styles

Discussion
This method uses a date formatter configured with the current default settings. The returned string is the same as if you configured and used a date formatter as shown in the following example:

```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setFormatterBehavior:NSDateFormatterBehavior10_4];
[formatter setDateStyle:dateStyle];
[formatter setTimeStyle:timeStyle];
NSString *result = [formatter stringForObjectValue:date];
```

Availability
Available in iOS 4.0 and later.

See Also
- `stringFromDate:` (page 353)

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 358).

Availability
Available in iOS 2.0 and later.

See Also
- `+ defaultFormatterBehavior` (page 326)
- `- formatterBehavior` (page 331)
- `- setFormatterBehavior:` (page 338)

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 iOS 2.0 and later.

See Also
- `setAMSymbol:` (page 335)
- `PMSymbol` (page 334)
- `setPMSymbol:` (page 341)

Declared In
`NSDateFormatter.h`

calendar

Returns the calendar for the receiver.

```
- (NSCalendar *)calendar
```

Return Value
The calendar for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setCalendar:` (page 336)

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 iOS 2.0 and later.

See Also
- `setDateFormat:` (page 336)

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 iOS 2.0 and later.

See Also

- `getObjectValue:forString:range:error:` (page 331)
- `stringFromDate:` (page 353)

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 357).

Availability

Available in iOS 2.0 and later.

See Also

- `setDateStyle:` (page 336)

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 iOS 2.0 and later.

See Also
- `setDefaultDate:` (page 337)

Declared In
`NSDateFormatter.h`

doesRelativeDateFormatting

Returns a Boolean value that indicates whether the receiver uses phrases such as “today” and “tomorrow” for the date component.

```
- (BOOL)doesRelativeDateFormatting
```

Return Value
`YES` if the receiver uses relative date formatting, otherwise `NO`.

Discussion
For a full discussion, see `setDoesRelativeDateFormatting:` (page 337).

Availability
Available in iOS 4.0 and later.

See Also
- `setDoesRelativeDateFormatting:` (page 337)

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 iOS 2.0 and later.

See Also
- `setEraSymbols:` (page 338)
- `longEraSymbols` (page 334)

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 358).

Availability
Available in iOS 2.0 and later.

See Also
`+ defaultFormatterBehavior` (page 326).
`+ setDefaultFormatterBehavior:` (page 327)
`- setFormatterBehavior:` (page 338)

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 iOS 2.0 and later.

See Also
`- setGeneratesCalendarDates:` (page 339)

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:(out id *)obj forString:(NSString *)string range:(inout
    NSRange *)rangep error:(out 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 iOS 2.0 and later.

See Also
- dateFromString: (page 329)
- stringForObjectValue: (page 588)

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 iOS 2.0 and later.

See Also
- setGregorianStartDate: (page 339)

Declared In
NSDateFormatter.h

init

Initializes and returns an NSDateFormatter instance. (Available in iOS 2.0 through iOS 3.2.)

```
- (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 336), `setTimeStyle:` (page 346), and appropriate style constants (see `NSDateFormatterStyle` (page 357)—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 338) message with the argument `NSDateFormatterBehavior10_4`.

Availability
Available in iOS 2.0 through iOS 3.2.

See Also
- `setDateStyle:` (page 336)
- `setTimeStyle:` (page 346)

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 iOS 2.0 and later.

See Also
- `setLenient:` (page 339)

Declared In
`NSDateFormatter.h`

locale

Returns the locale for the receiver.

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

Return Value
The locale for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setLocale:` (page 340)

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 iOS 2.0 and later.

See Also
- setLongEraSymbols: (page 340)
- eraSymbols (page 330)

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 iOS 2.0 and later.

See Also
- setMonthSymbols: (page 341)
- shortMonthSymbols (page 349)
- veryShortMonthSymbols (page 355)
- standaloneMonthSymbols (page 352)
- shortStandaloneMonthSymbols (page 350)
- veryShortStandaloneMonthSymbols (page 355)

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 iOS 2.0 and later.

See Also
- `setPMSymbol:` (page 341)
- `AMSymbol` (page 327)
- `setAMSymbol:` (page 335)

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 iOS 2.0 and later.

See Also
- `setQuarterSymbols:` (page 341)
- `shortQuarterSymbols` (page 350)
- `standaloneQuarterSymbols` (page 352)
- `shortStandaloneQuarterSymbols` (page 351)

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 iOS 2.0 and later.

See Also
- `AMSymbol` (page 327)
- `PMSymbol` (page 334)
- `setPMSymbol:` (page 341)

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 iOS 2.0 and later.

See Also
- calendar (page 328)

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 Guide* for a list of the conversion specifiers permitted in date format strings.

Availability
Available in iOS 2.0 and later.

See Also
- dateFormat (page 328).

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 357).

Availability
Available in iOS 2.0 and later.

See Also
- `dateStyle` (page 329).

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 iOS 2.0 and later.

See Also
- `defaultDate` (page 329)

Declared In
`NSDateFormatter.h`

setDoesRelativeDateFormatting:

Specifies whether the receiver uses phrases such as “today” and “tomorrow” for the date component.

```
- (void)setDoesRelativeDateFormatting:(BOOL)b
```

Parameters

b

`YES` to specify that the receiver should use relative date formatting, otherwise `NO`.

Discussion
If a date formatter uses relative date formatting, where possible it replaces the date component of its output with a phrase—such as “today” or “tomorrow”—that indicates a relative date. The available phrases depend on the locale for the date formatter; whereas, for dates in the future, English may only allow “tomorrow,” French may allow “the day after the day after tomorrow,” as illustrated in the following example.

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
[dateFormatter setTimeStyle:NSDateFormatterNoStyle];
[dateFormatter setDateStyle:NSDateFormatterMediumStyle];
NSLocale *frLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"fr_FR"];
[dateFormatter setLocale:frLocale];

[dateFormatter setDoesRelativeDateFormatting:YES];

NSDate *date = [NSDate dateWithTimeIntervalSinceNow:60*60*24*3];
NSString *dateString = [dateFormatter stringFromDate:date];
```

```
NSLog(@"dateString: %@", dateString);
// Output
// dateString: après-après-demain
```

Availability
Available in iOS 4.0 and later.

See Also
- `doesRelativeDateFormatting` (page 330)

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 iOS 2.0 and later.

See Also
- `eraSymbols` (page 330)
- `longEraSymbols` (page 334)

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 358).

Availability
Available in iOS 2.0 and later.

See Also
- `+ defaultFormatterBehavior` (page 326).
- `+ setDefaultFormatterBehavior:` (page 327)
- `- formatterBehavior` (page 331)

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 iOS 2.0 and later.

See Also
- generatesCalendarDates (page 331).

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 iOS 2.0 and later.

See Also
- gregorianStartDate (page 332)

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 iOS 2.0 and later.

See Also
- `isLenient` (page 333)

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 iOS 2.0 and later.

See Also
- `locale` (page 333)

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 iOS 2.0 and later.

See Also
- `longEraSymbols` (page 334)
- `eraSymbols` (page 330)

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 iOS 2.0 and later.

See Also
- `monthSymbols` (page 334)
- `setShortMonthSymbols:` (page 342)
- `setVeryShortMonthSymbols:` (page 347)
- `setStandaloneMonthSymbols:` (page 345)
- `setShortStandaloneMonthSymbols:` (page 343)
- `setVeryShortStandaloneMonthSymbols:` (page 347)

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 iOS 2.0 and later.

See Also
- `PMSymbol` (page 334)
- `AMSymbol` (page 327)
- `setAMSymbol:` (page 335)

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 iOS 2.0 and later.

See Also
- `quarterSymbols` (page 335)
- `setShortQuarterSymbols:` (page 342)
- `setStandaloneQuarterSymbols:` (page 345)
- `setShortStandaloneQuarterSymbols:` (page 343)

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 iOS 2.0 and later.

See Also
- `shortMonthSymbols` (page 349)
- `setMonthSymbols:` (page 341)
- `setVeryShortMonthSymbols:` (page 347)
- `setStandaloneMonthSymbols:` (page 345)
- `setShortStandaloneMonthSymbols:` (page 343)
- `setVeryShortStandaloneMonthSymbols:` (page 347)

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 iOS 2.0 and later.

See Also
- `shortQuarterSymbols` (page 350)
- `setQuarterSymbols:` (page 341)
- `setStandaloneQuarterSymbols:` (page 345)
- `setShortStandaloneQuarterSymbols:` (page 343)

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 iOS 2.0 and later.

See Also
- `shortStandaloneMonthSymbols` (page 350)
- `setMonthSymbols:` (page 341)
- `setShortMonthSymbols:` (page 342)
- `setVeryShortMonthSymbols:` (page 347)
- `setStandaloneMonthSymbols:` (page 345)
- `setVeryShortStandaloneMonthSymbols:` (page 347)

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 iOS 2.0 and later.

See Also
- `shortStandaloneQuarterSymbols` (page 351)
- `setQuarterSymbols:` (page 341)
- `setShortQuarterSymbols:` (page 342)
- `setStandaloneQuarterSymbols:` (page 345)

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 iOS 2.0 and later.

See Also
- shortStandaloneWeekdaySymbols (page 351)
- setWeekdaySymbols: (page 349)
- setShortWeekdaySymbols: (page 344)
- setVeryShortWeekdaySymbols: (page 348)
- setStandaloneWeekdaySymbols: (page 345)
- setVeryShortStandaloneWeekdaySymbols: (page 348)

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 iOS 2.0 and later.

See Also
- shortWeekdaySymbols (page 352)
- setWeekdaySymbols: (page 349)
- setVeryShortWeekdaySymbols: (page 348)
- setStandaloneWeekdaySymbols: (page 345)
- setShortStandaloneWeekdaySymbols: (page 344)
- setVeryShortStandaloneWeekdaySymbols: (page 348)

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 iOS 2.0 and later.

See Also
- `standaloneMonthSymbols` (page 352)
- `setMonthSymbols:` (page 341)
- `setShortMonthSymbols:` (page 342)
- `setVeryShortMonthSymbols:` (page 347)
- `setShortStandaloneMonthSymbols:` (page 343)
- `setVeryShortStandaloneMonthSymbols:` (page 347)

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 iOS 2.0 and later.

See Also
- `setStandaloneQuarterSymbols:` (page 345)
- `setQuarterSymbols:` (page 341)
- `setShortQuarterSymbols:` (page 342)
- `setShortStandaloneQuarterSymbols:` (page 343)

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 iOS 2.0 and later.

See Also
- `standaloneWeekdaySymbols` (page 353)
- `setWeekdaySymbols:` (page 349)
- `setShortWeekdaySymbols:` (page 344)
- `setVeryShortWeekdaySymbols:` (page 348)
- `setShortStandaloneWeekdaySymbols:` (page 344)
- `setVeryShortStandaloneWeekdaySymbols:` (page 348)

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 357).

Availability
Available in iOS 2.0 and later.

See Also
- `timeStyle` (page 354)

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 iOS 2.0 and later.

See Also
- `timeZone` (page 354)

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 iOS 2.0 and later.

See Also
- twoDigitStartDate (page 354)

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 iOS 2.0 and later.

See Also
- veryShortMonthSymbols (page 355)
- setMonthSymbols: (page 341)
- setShortMonthSymbols: (page 342)
- setStandaloneMonthSymbols: (page 345)
- setShortStandaloneMonthSymbols: (page 343)
- setVeryShortStandaloneMonthSymbols: (page 347)

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 iOS 2.0 and later.

See Also
- `veryShortStandaloneMonthSymbols` (page 355)
- `setMonthSymbols:` (page 341)
- `setShortMonthSymbols:` (page 342)
- `setVeryShortMonthSymbols:` (page 347)
- `setStandaloneMonthSymbols:` (page 345)
- `setShortStandaloneMonthSymbols:` (page 343)

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 iOS 2.0 and later.

See Also
- `veryShortStandaloneWeekdaySymbols` (page 356)
- `setWeekdaySymbols:` (page 349)
- `setShortWeekdaySymbols:` (page 344)
- `setVeryShortWeekdaySymbols:` (page 348)
- `setStandaloneWeekdaySymbols:` (page 345)
- `setShortStandaloneWeekdaySymbols:` (page 344)

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 iOS 2.0 and later.

See Also
- `veryShortWeekdaySymbols` (page 356)
- `setWeekdaySymbols:` (page 349)
- `setShortWeekdaySymbols:` (page 344)
- `setStandaloneWeekdaySymbols:` (page 345)
- `setShortStandaloneWeekdaySymbols:` (page 344)
- `setVeryShortStandaloneWeekdaySymbols:` (page 348)

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 iOS 2.0 and later.

See Also
- `weekdaySymbols` (page 357)
- `setShortWeekdaySymbols:` (page 344)
- `setVeryShortWeekdaySymbols:` (page 348)
- `setStandaloneWeekdaySymbols:` (page 345)
- `setShortStandaloneWeekdaySymbols:` (page 344)
- `setVeryShortStandaloneWeekdaySymbols:` (page 348)

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 iOS 2.0 and later.

See Also
- `setShortMonthSymbols:` (page 342)
- `monthSymbols` (page 334)
- `veryShortMonthSymbols` (page 355)
- `standaloneMonthSymbols` (page 352)
- `shortStandaloneMonthSymbols` (page 350)
- `veryShortStandaloneMonthSymbols` (page 355)

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 iOS 2.0 and later.

See Also
- `setShortQuarterSymbols:` (page 342)
- `quarterSymbols` (page 335)
- `standaloneQuarterSymbols` (page 352)
- `shortStandaloneQuarterSymbols` (page 351)

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 iOS 2.0 and later.

See Also
- `setShortStandaloneMonthSymbols:` (page 343)
- `monthSymbols` (page 334)
- `shortMonthSymbols` (page 349)
- `veryShortMonthSymbols` (page 355)
- `standaloneMonthSymbols` (page 352)

- `veryShortStandaloneMonthSymbols` (page 355)

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 iOS 2.0 and later.

See Also
- `setShortStandaloneQuarterSymbols:` (page 343)
- `quarterSymbols` (page 335)
- `shortQuarterSymbols` (page 350)
- `standaloneQuarterSymbols` (page 352)

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 iOS 2.0 and later.

See Also
- `setShortStandaloneWeekdaySymbols:` (page 344)
- `weekdaySymbols` (page 357)
- `shortWeekdaySymbols` (page 352)
- `veryShortWeekdaySymbols` (page 356)
- `standaloneWeekdaySymbols` (page 353)
- `veryShortStandaloneWeekdaySymbols` (page 356)

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 iOS 2.0 and later.

See Also
- `setShortWeekdaySymbols:` (page 344)
- `weekdaySymbols` (page 357)
- `veryShortWeekdaySymbols` (page 356)
- `standaloneWeekdaySymbols` (page 353)
- `shortStandaloneWeekdaySymbols` (page 351)
- `veryShortStandaloneWeekdaySymbols` (page 356)

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 iOS 2.0 and later.

See Also
- `monthSymbols` (page 334)
- `setStandaloneMonthSymbols:` (page 345)
- `shortMonthSymbols` (page 349)
- `veryShortMonthSymbols` (page 355)
- `shortStandaloneMonthSymbols` (page 350)
- `veryShortStandaloneMonthSymbols` (page 355)

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 iOS 2.0 and later.

See Also
- `setStandaloneQuarterSymbols:` (page 345)
- `quarterSymbols` (page 335)
- `shortQuarterSymbols` (page 350)
- `shortStandaloneQuarterSymbols` (page 351)

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 iOS 2.0 and later.

See Also
- `setStandaloneWeekdaySymbols:` (page 345)
- `weekdaySymbols` (page 357)
- `shortWeekdaySymbols` (page 352)
- `veryShortWeekdaySymbols` (page 356)
- `shortStandaloneWeekdaySymbols` (page 351)
- `veryShortStandaloneWeekdaySymbols` (page 356)

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 iOS 2.0 and later.

See Also
- `dateFromString:` (page 329)
+ `localizedStringFromDate:dateStyle:timeStyle:` (page 326)

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 357).

Availability
Available in iOS 2.0 and later.

See Also
- `setTimeStyle:` (page 346)

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 iOS 2.0 and later.

See Also
- `setTimeZone:` (page 346)

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 iOS 2.0 and later.

See Also
- `setTwoDigitStartDate:` (page 347)

Declared In
`NSDateFormatter.h`

verySho rtMonthSymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)verySho rtMonthSymbols
```

Return Value
An array of `NSString` objects that specify the very short month symbols for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setVerySho rtMonthSymbols:` (page 347)
- `monthSymbols` (page 334)
- `shortMonthSymbols` (page 349)
- `standaloneMonthSymbols` (page 352)
- `shortStandaloneMonthSymbols` (page 350)
- `verySho rtStandaloneMonthSymbols` (page 355)

Declared In
`NSDateFormatter.h`

verySho rtStandaloneMonthSymbols

Returns the very short month symbols for the receiver.

```
- (NSArray *)verySho rtStandaloneMonthSymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone month symbols for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setVeryShortStandaloneMonthSymbols:` (page 347)
- `monthSymbols` (page 334)
- `shortMonthSymbols` (page 349)
- `veryShortMonthSymbols` (page 355)
- `standaloneMonthSymbols` (page 352)
- `shortStandaloneMonthSymbols` (page 350)

Declared In
`NSDateFormatter.h`

verySymbolStandaloneWeekdaySymbols

Returns the array of very short standalone weekday symbols for the receiver.

```
- (NSArray *)verySymbolStandaloneWeekdaySymbols
```

Return Value
An array of `NSString` objects that specify the very short standalone weekday symbols for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `setShortStandaloneWeekdaySymbols:` (page 344)
- `weekdaySymbols` (page 357)
- `shortWeekdaySymbols` (page 352)
- `veryShortWeekdaySymbols` (page 356)
- `standaloneWeekdaySymbols` (page 353)
- `shortStandaloneWeekdaySymbols` (page 351)

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 iOS 2.0 and later.

See Also
- `setVeryShortWeekdaySymbols:` (page 348)
- `weekdaySymbols` (page 357)
- `shortWeekdaySymbols` (page 352)

- standaloneWeekdaySymbols (page 353)
- shortStandaloneWeekdaySymbols (page 351)
- veryShortStandaloneWeekdaySymbols (page 356)

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 iOS 2.0 and later.

See Also
- setWeekdaySymbols: (page 349)
- shortWeekdaySymbols (page 352)
- veryShortWeekdaySymbols (page 356)
- standaloneWeekdaySymbols (page 353)
- shortStandaloneWeekdaySymbols (page 351)
- veryShortStandaloneWeekdaySymbols (page 356)

Declared In
NSDateFormatter.h

Constants

NSDateFormatterStyle

The following constants specify predefined format styles for dates and times.

```
typedef enum {
    NSDateFormatterNoStyle     = kCFDateFormatterNoStyle,
    NSDateFormatterShortStyle  = kCFDateFormatterShortStyle,
    NSDateFormatterMediumStyle = kCFDateFormatterMediumStyle,
    NSDateFormatterLongStyle   = kCFDateFormatterLongStyle,
    NSDateFormatterFullStyle   = kCFDateFormatterFullStyle
} NSDateFormatterStyle;
```

Constants

`NSDateFormatterNoStyle`

Specifies no style.

Equal to `kCFDateFormatterNoStyle`.

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSDateFormatter.h`.

`NSDateFormatterBehavior10_0`

Specifies formatting behavior equivalent to that in Mac OS X 10.0.

Available in iOS 2.0 through iOS 2.1.

Declared in `NSDateFormatter.h`.

`NSDateFormatterBehavior10_4`

Specifies formatting behavior equivalent for Mac OS X 10.4.

Available in iOS 2.0 and later.

Declared in `NSDateFormatter.h`.

Availability

Available in iOS 2.0 and later.

Declared In

`NSDateFormatter.h`

NSDecimalNumber Class Reference

Inherits from	NSNumber : NSValue : NSObject
Conforms to	NSCoding (NSValue) NSCopying (NSValue) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics

Overview

`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 364)
Creates and returns an `NSDecimalNumber` object equivalent to a given `NSDecimal` structure.

`+ decimalNumberWithMantissa:exponent:isNegative:` (page 364)
Creates and returns an `NSDecimalNumber` object equivalent to the number specified by the arguments.

`+ decimalNumberWithString:` (page 365)
Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string.

`+ decimalNumberWithString:locale:` (page 365)
Creates and returns an `NSDecimalNumber` object whose value is equivalent to that in a given numeric string, interpreted using a given locale.

`+ one` (page 368)
Returns an `NSDecimalNumber` object equivalent to the number 1.0.

+ zero (page 369)
Returns an NSDecimalNumber object equivalent to the number 0.0.

+ notANumber (page 367)
Returns an NSDecimalNumber object that specifies no number.

Initializing a Decimal Number

- initWithDecimal: (page 375)
Returns an NSDecimalNumber object initialized to represent a given decimal.

- initWithMantissa:exponent:isNegative: (page 376)
Returns an NSDecimalNumber object initialized using the given mantissa, exponent, and sign.

- initWithString: (page 377)
Returns an NSDecimalNumber object initialized so that its value is equivalent to that in a given numeric string.

- initWithString:locale: (page 377)
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 369)
Returns a new NSDecimalNumber object whose value is the sum of the receiver and another given NSDecimalNumber object.

- decimalNumberBySubtracting: (page 374)
Returns a new NSDecimalNumber object whose value is that of another given NSDecimalNumber object subtracted from the value of the receiver.

- decimalNumberByMultiplyingBy: (page 371)
Returns a new NSDecimalNumber object whose value is the value of the receiver multiplied by that of another given NSDecimalNumber object.

- decimalNumberByDividingBy: (page 370)
Returns a new NSDecimalNumber object whose value is the value of the receiver divided by that of another given NSDecimalNumber object.

- decimalNumberByRaisingToPower: (page 373)
Returns a new NSDecimalNumber object whose value is the value of the receiver raised to a given power.

- decimalNumberByMultiplyingByPowerOf10: (page 372)
Multiplies the receiver by 10^*power* and returns the product, a newly created NSDecimalNumber object.

- decimalNumberByAdding:withBehavior: (page 370)
Adds *decimalNumber* to the receiver and returns the sum, a newly created NSDecimalNumber object.

- decimalNumberBySubtracting:withBehavior: (page 374)
Subtracts *decimalNumber* from the receiver and returns the difference, a newly created NSDecimalNumber object.

- `decimalNumberByMultiplyingBy:withBehavior:` (page 372)
  Multiplies the receiver by *`decimalNumber`* and returns the product, a newly created `NSDecimalNumber` object.
- `decimalNumberByDividingBy:withBehavior:` (page 371)
  Divides the receiver by *`decimalNumber`* and returns the quotient, a newly created `NSDecimalNumber` object.
- `decimalNumberByRaisingToPower:withBehavior:` (page 373)
  Raises the receiver to *`power`* and returns the result, a newly created `NSDecimalNumber` object.
- `decimalNumberByMultiplyingByPowerOf10:withBehavior:` (page 372)
  Multiplies the receiver by 10^*`power`* and returns the product, a newly created `NSDecimalNumber` object.

Rounding Off

- `decimalNumberByRoundingAccordingToBehavior:` (page 373)
  Rounds the receiver off in the way specified by *`behavior`* and returns the result, a newly created `NSDecimalNumber` object.

Accessing the Value

- `decimalValue` (page 375)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 375)
  Returns the approximate value of the receiver as a `double`.
- `descriptionWithLocale:` (page 375)
  Returns a string, specified according to a given locale, that represents the contents of the receiver.
- `objCType` (page 378)
  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 366)
  Returns the way arithmetic methods, like `decimalNumberByAdding:` (page 369), round off and handle error conditions.
+ `setDefaultBehavior:` (page 368)
  Specifies the way that arithmetic methods, like `decimalNumberByAdding:` (page 369), round off and handle error conditions.

Comparing Decimal Numbers

- `compare:` (page 369)
  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 367)
Returns the largest possible value of an `NSDecimalNumber` object.

`+ minimumDecimalNumber` (page 367)
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 1121)

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
`+ decimalNumberWithString:locale:` (page 365)

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 iOS 2.0 and later.

See Also

`+ decimalNumberWithString:` (page 365)

Declared In

`NSDecimalNumber.h`

defaultBehavior

Returns the way arithmetic methods, like `decimalNumberByAdding:` (page 369), 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 370). If you find yourself using a particular behavior consistently, you can specify a different default behavior with `setDefaultBehavior:` (page 368).

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`+ minimumDecimalNumber` (page 367)

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 iOS 2.0 and later.

See Also
`+ maximumDecimalNumber` (page 367)

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 1556) method description in the `NSDecimalNumberBehaviors` protocol specification.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`+ zero` (page 369)

Declared In
`NSDecimalNumber.h`

setDefaultBehavior:

Specifies the way that arithmetic methods, like `decimalNumberByAdding:` (page 369), round off and handle error conditions.

```
+ (void)setDefaultBehavior:(id < NSDecimalNumberBehaviors >)behavior
```

Discussion
`behavior` must conform to the `NSDecimalNumberBehaviors` protocol.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`+ one` (page 368)

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 iOS 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 iOS 2.0 and later.

See Also

`- decimalNumberByAdding:withBehavior:` (page 370)

`+ defaultBehavior` (page 366)

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 iOS 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 iOS 2.0 and later.

See Also
- `decimalNumberByDividingBy:withBehavior:` (page 371)
+ `defaultBehavior` (page 366)

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 iOS 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 iOS 2.0 and later.

See Also
- `decimalNumberByMultiplyingBy:withBehavior:` (page 372)
+ `defaultBehavior` (page 366)

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 iOS 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 iOS 2.0 and later.

See Also
- decimalNumberByMultiplyingByPowerOf10:withBehavior: (page 372)
+ defaultBehavior (page 366)

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 iOS 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 iOS 2.0 and later.

See Also

- `decimalNumberByRaisingToPower:withBehavior:` (page 373)
+ `defaultBehavior` (page 366)

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 iOS 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 911) method in the `NSDecimalNumberBehaviors` protocol specification.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- decimalNumberBySubtracting:withBehavior: (page 374)
+ defaultBehavior (page 366)

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also

`+ decimalNumberWithMantissa:exponent:isNegative:` (page 364)

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 365).

Return Value

An `NSDecimalNumber` object initialized so that its value is equivalent to that in *numericString*.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`+ decimalNumberWithString:locale:` (page 365)

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 iOS 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 iOS 2.0 and later.
Declared in `NSDecimalNumber.h`.

`NSDecimalNumberOverflowException`
The name of the exception raised on overflow.
Available in iOS 2.0 and later.
Declared in `NSDecimalNumber.h`.

`NSDecimalNumberUnderflowException`
The name of the exception raised on underflow.
Available in iOS 2.0 and later.
Declared in `NSDecimalNumber.h`.

`NSDecimalNumberDivideByZeroException`
The name of the exception raised on divide by zero.
Available in iOS 2.0 and later.
Declared in `NSDecimalNumber.h`.

Declared In
`NSDecimalNumber.h`

NSDecimalNumberHandler Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSDecimalNumberBehaviors NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics

Overview

`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 1556)
- `scale` (page 1557)
- `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1556)

NSCoding
- `encodeWithCoder:` (page 1552)
- `initWithCoder:` (page 1552)

Tasks

Creating a Decimal Number Handler

`+ defaultDecimalNumberHandler` (page 381)
Returns the default instance of `NSDecimalNumberHandler`.

`+ decimalNumberHandlerWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:` (page 380)
Returns an `NSDecimalNumberHandler` object with customized behavior.

Initializing a Decimal Number Handler

`- initWithRoundingMode:scale:raiseOnExactness:raiseOnOverflow:raiseOnUnderflow:raiseOnDivideByZero:` (page 381)
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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In

`NSDecimalNumber.h`

NSDictionary Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDictionary.h Foundation/NSFileManager.h Foundation/NSKeyValueCoding.h
Companion guides	Collections Programming Topics Property List Programming Guide
Related sample code	BonjourWeb KeyboardAccessory MoviePlayer SpeakHere WiTap

Overview

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 1632)). 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 1638) 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 410) and `objectEnumerator` (page 413) 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]);
}
```

On Mac OS X v10.6 and later, `NSDictionary` supports enumeration using block objects.

Primitive Methods

Three primitive methods of `NSDictionary`—`count` (page 395), `objectForKey:` (page 414), and `keyEnumerator` (page 410)—provide the basis for all of the other methods in its interface. The count (page 395) method returns the number of entries in the dictionary. objectForKey: (page 414) returns the value associated with a given key. keyEnumerator (page 410) 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 415) 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*.

Toll-Free Bridging

`NSDictionary` is "toll-free bridged" with its Core Foundation counterpart, *CFDictionary 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 `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.

Subclassing

There should typically be little need to subclass `NSDictionary`. If you do need to customize behavior, it is often better to consider composition rather than subclassing.

If you do need to subclass `NSDictionary`, you need to take into account that is represented by a Class cluster—there are therefore several primitive methods upon which the methods are conceptually based:

- `count` (page 395)
- `objectForKey:` (page 414)
- `keyEnumerator` (page 410)

In a subclass, you must override all these methods.

`NSDictionary`'s other methods operate by invoking one or more of these primitives. The non-primitive methods provide convenient ways of accessing multiple entries at once.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

NSMutableCopying
- `- mutableCopyWithZone:` (page 1614)

NSFastEnumeration
- `- countByEnumeratingWithState:objects:count:` (page 1569)

Tasks

Creating a Dictionary

`+ dictionary` (page 389)
Creates and returns an empty dictionary.

`+ dictionaryWithContentsOfFile:` (page 390)
Creates and returns a dictionary using the keys and values found in a file specified by a given path.

`+ dictionaryWithContentsOfURL:` (page 390)
Creates and returns a dictionary using the keys and values found in a resource specified by a given URL.

`+ dictionaryWithDictionary:` (page 391)
Creates and returns a dictionary containing the keys and values from another given dictionary.

`+ dictionaryWithObject:forKey:` (page 391)
Creates and returns a dictionary containing a given key and value.

`+ dictionaryWithObjects:forKeys:` (page 392)
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 392)
Creates and returns a dictionary containing *`count`* objects from the *`objects`* array.

`+ dictionaryWithObjectsAndKeys:` (page 393)
Creates and returns a dictionary containing entries constructed from the specified set of values and keys.

Initializing an NSDictionary Instance

`- initWithContentsOfFile:` (page 405)
Initializes a newly allocated dictionary using the keys and values found in a file at a given path.

`- initWithContentsOfURL:` (page 406)
Initializes a newly allocated dictionary using the keys and values found at a given URL.

`- initWithDictionary:` (page 406)
Initializes a newly allocated dictionary by placing in it the keys and values contained in another given dictionary.

`- initWithDictionary:copyItems:` (page 407)
Initializes a newly allocated dictionary using the objects contained in another given dictionary.

`- initWithObjects:forKeys:` (page 407)
Initializes a newly allocated dictionary with entries constructed from the contents of the *`objects`* and *`keys`* arrays.

`- initWithObjects:forKeys:count:` (page 408)
Initializes a newly allocated dictionary with *`count`* entries.

`- initWithObjectsAndKeys:` (page 409)
Initializes a newly allocated dictionary with entries constructed from the specified set of values and keys.

Counting Entries

- `count` (page 395)
  Returns the number of entries in the receiver.

Comparing Dictionaries

- `isEqualToDictionary:` (page 409)
  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 394)
  Returns a new array containing the receiver's keys.
- `allKeysForObject:` (page 394)
  Returns a new array containing the keys corresponding to all occurrences of a given object in the receiver.
- `allValues` (page 395)
  Returns a new array containing the receiver's values.
- `getObjects:andKeys:` (page 405)
  Returns by reference C arrays of the keys and values in the receiver.
- `objectForKey:` (page 414)
  Returns the value associated with a given key.
- `objectsForKeys:notFoundMarker:` (page 414)
  Returns the set of objects from the receiver that corresponds to the specified *`keys`* as an NSArray.
- `valueForKey:` (page 415)
  Returns the value associated with a given key.

Enumerating Dictionaries

- `keyEnumerator` (page 410)
  Returns an enumerator object that lets you access each key in the receiver.
- `objectEnumerator` (page 413)
  Returns an enumerator object that lets you access each value in the receiver.
- `enumerateKeysAndObjectsUsingBlock:` (page 398)
  Applies a given block object to the entries of the receiver.
- `enumerateKeysAndObjectsWithOptions:usingBlock:` (page 398)
  Applies a given block object to the entries of the receiver.

Sorting Dictionaries

- `keysSortedByValueUsingSelector:` (page 412)
  Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values.
- `keysSortedByValueUsingComparator:` (page 411)
  Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using a given comparator block.
- `keysSortedByValueWithOptions:usingComparator:` (page 412)
  Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using a given comparator block and a specified set of options.

Filtering Dictionaries

- `keysOfEntriesPassingTest:` (page 410)
  Returns the set of keys whose corresponding value satisfies a constraint described by a block object.
- `keysOfEntriesWithOptions:passingTest:` (page 411)
  Returns the set of keys whose corresponding value satisfies a constraint described by a block object.

Storing Dictionaries

- `writeToFile:atomically:` (page 415)
  Writes a property list representation of the contents of the receiver to a given path.
- `writeToURL:atomically:` (page 416)
  Writes a property list representation of the contents of the receiver to a given URL.

Accessing File Attributes

- `fileCreationDate` (page 399)
  Returns the value for the `NSFileCreationDate` key.
- `fileExtensionHidden` (page 399)
  Returns the value for the `NSFileExtensionHidden` key.
- `fileGroupOwnerAccountID` (page 399)
  Returns the value for the `NSFileGroupOwnerAccountID` key.
- `fileGroupOwnerAccountName` (page 400)
  Returns the value for the `NSFileGroupOwnerAccountName` key.
- `fileHFSCreatorCode` (page 400)
  Returns the value for the `NSFileHFSCreatorCode` key.
- `fileHFSTypeCode` (page 401)
  Returns the value for the `NSFileHFSTypeCode` key.
- `fileIsAppendOnly` (page 401)
  Returns the value for the `NSFileAppendOnly` key.

- `fileIsImmutable` (page 401)
  Returns the value for the `NSFileImmutable` key.
- `fileModificationDate` (page 402)
  Returns the value for the key `NSFileModificationDate`.
- `fileOwnerAccountID` (page 402)
  Returns the value for the `NSFileOwnerAccountID` key.
- `fileOwnerAccountName` (page 402)
  Returns the value for the key `NSFileOwnerAccountName`.
- `filePosixPermissions` (page 403)
  Returns the value for the key `NSFilePosixPermissions`.
- `fileSize` (page 403)
  Returns the value for the key `NSFileSize`.
- `fileSystemFileNumber` (page 404)
  Returns the value for the key `NSFileSystemFileNumber`.
- `fileSystemNumber` (page 404)
  Returns the value for the key `NSFileSystemNumber`.
- `fileType` (page 405)
  Returns the value for the key `NSFileType`.

Creating a Description

- `description` (page 396)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionInStringsFileFormat` (page 396)
  Returns a string that represents the contents of the receiver, formatted in `.strings` file format.
- `descriptionWithLocale:` (page 397)
  Returns a string object that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 397)
  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 iOS 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*.

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 iOS 2.0 and later.

See Also
`- initWithContentsOfFile:` (page 405)

Related Sample Code
MoviePlayer

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*.

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 iOS 2.0 and later.

See Also
- `initWithContentsOfURL:` (page 406)

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 iOS 2.0 and later.

See Also
- `initWithDictionary:` (page 406)

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 iOS 2.0 and later.

See Also
- `+ dictionaryWithObjects:forKeys:` (page 392)
- `+ dictionaryWithObjects:forKeys:count:` (page 392)
- `+ dictionaryWithObjectsAndKeys:` (page 393)

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.

keys
An array containing the keys for the new dictionary. Each key is copied (using copyWithZone: (page 1554); 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 iOS 2.0 and later.

See Also
- initWithObjects:forKeys: (page 407)
+ dictionaryWithObject:forKey: (page 391)
+ dictionaryWithObjects:forKeys:count: (page 392)
+ dictionaryWithObjectsAndKeys: (page 393)

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.

keys
A C array of keys for the new dictionary. Each key is copied (using copyWithZone: (page 1554); 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 iOS 2.0 and later.

See Also
- `initWithObjects:forKeys:count:` (page 408)
+ `dictionaryWithObject:forKey:` (page 391)
+ `dictionaryWithObjects:forKeys:` (page 392)
+ `dictionaryWithObjectsAndKeys:` (page 393)

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 392), differing only in the way key-value pairs are specified.

For example:

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"value1", @"key1", @"value2", @"key2", nil];
```

Availability
Available in iOS 2.0 and later.

See Also
- `initWithObjectsAndKeys:` (page 409)
+ `dictionaryWithObject:forKey:` (page 391)
+ `dictionaryWithObjects:forKeys:` (page 392)
+ `dictionaryWithObjects:forKeys:count:` (page 392)

Related Sample Code
aurioTouch
GLSprite
MoviePlayer
SpeakHere

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 iOS 2.0 and later.

See Also
- `allValues` (page 395)
- `allKeysForObject:` (page 394)
- `getObjects:andKeys:` (page 405)

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 1632) message to determine if it's equal to *anObject*.

Availability

Available in iOS 2.0 and later.

See Also

- `allKeys` (page 394)
- `keyEnumerator` (page 410)

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 iOS 2.0 and later.

See Also

- `allKeys` (page 394)
- `getObjects:andKeys:` (page 405)
- `objectEnumerator` (page 413)

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 iOS 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. This method is intended to produce readable output for debugging purposes, not for serializing data. If you want to store dictionary data for later retrieval, see *Property List Programming Guide* and *Archives and Serializations Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
- `descriptionWithLocale:` (page 397)
- `descriptionWithLocale:indent:` (page 397)

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 iOS 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 397).

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 iOS 2.0 and later.

See Also

- `description` (page 396)
- `descriptionWithLocale:indent:` (page 397)

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 iOS 2.0 and later.

See Also
- `description` (page 396)
- `descriptionWithLocale:` (page 397)

Declared In
`NSDictionary.h`

enumerateKeysAndObjectsUsingBlock:

Applies a given block object to the entries of the receiver.

```
- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block
```

Parameters

block
A block object to operate on entries in the receiver.

Discussion
If the block sets `*stop` to `YES`, the enumeration stops.

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateKeysAndObjectsWithOptions:usingBlock:` (page 398)

Declared In
`NSDictionary.h`

enumerateKeysAndObjectsWithOptions:usingBlock:

Applies a given block object to the entries of the receiver.

```
- (void)enumerateKeysAndObjectsWithOptions:(NSEnumerationOptions)opts
    usingBlock:(void (^)(id key, id obj, BOOL *stop))block
```

Parameters

opts

Enumeration options.

block

A block object to operate on entries in the receiver.

Discussion

If the block sets `*stop` to `YES`, the enumeration stops.

Availability

Available in iOS 4.0 and later.

See Also

`- enumerateKeysAndObjectsUsingBlock:` (page 398)

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 iOS 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 iOS 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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 iOS 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 iOS 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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory.

Availability
Available in iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`NSDirectoryEnumerator`), that represents the POSIX attributes of a file or directory. This method returns the file's permissions.

Availability
Available in iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 515) (NSFileManager), directoryAttributes (page 420) (NSDirectoryEnumerator), and fileAttributes (page 420) (NSDirectoryEnumerator), that represents the POSIX attributes of a file or directory. This method returns the file's inode.

Availability
Available in iOS 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 515) (NSFileManager), directoryAttributes (page 420) (NSDirectoryEnumerator), and fileAttributes (page 420) (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 iOS 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 515) (`NSFileManager`), `directoryAttributes` (page 420) (`NSDirectoryEnumerator`), and `fileAttributes` (page 420) (`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 iOS 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 iOS 2.0 and later.

See Also
- `allKeys` (page 394)
- `allValues` (page 395)
- `objectForKey:` (page 414)
- `objectsForKeys:notFoundMarker:` (page 414)

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*.

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 iOS 2.0 and later.

See Also

`+ dictionaryWithContentsOfFile:` (page 390)

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*.

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 iOS 2.0 and later.

See Also

`+ dictionaryWithContentsOfURL:` (page 390)

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 iOS 2.0 and later.

See Also

+ dictionaryWithDictionary: (page 391)

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 1554) 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 iOS 2.0 and later.

See Also

- initWithDictionary: (page 406)

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.

keys

An array containing the keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 1554); 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 iOS 2.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:` (page 392)
`- initWithObjects:forKeys:count:` (page 408)
`- initWithObjectsAndKeys:` (page 409)

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.

keys

A C array of keys for the new dictionary. Each key is copied (using `copyWithZone:` (page 1554); 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 iOS 2.0 and later.

See Also

`+ dictionaryWithObjects:forKeys:count:` (page 392)
`- initWithObjects:forKeys:` (page 407)
`- initWithObjectsAndKeys:` (page 409)

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 407), 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 iOS 2.0 and later.

See Also
+ dictionaryWithObjectsAndKeys: (page 393)
- initWithObjects:forKeys: (page 407)
- initWithObjects:forKeys:count: (page 408)

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 1632) test.

Availability
Available in iOS 2.0 and later.

See Also
- `isEqual:` (page 1632) (`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 394) 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 413) method provides a convenient way to access each value in the dictionary.

Availability
Available in iOS 2.0 and later.

See Also
- `allKeys` (page 394)
- `allKeysForObject:` (page 394)
- `getObjects:andKeys:` (page 405)
- `objectEnumerator` (page 413)
- `nextObject` (page 424) (NSEnumerator)

Declared In
`NSDictionary.h`

keysOfEntriesPassingTest:

Returns the set of keys whose corresponding value satisfies a constraint described by a block object.

```
- (NSSet *)keysOfEntriesPassingTest:(BOOL (^)(id key, id obj, BOOL *stop))predicate
```

Parameters

predicate

A block object that specifies constraints for values in the receiver.

Return Value

The set of keys whose corresponding value satisfies *predicate*.

Availability

Available in iOS 4.0 and later.

See Also

- enumerateKeysAndObjectsUsingBlock: (page 398)

Declared In

NSDictionary.h

keysOfEntriesWithOptions:passingTest:

Returns the set of keys whose corresponding value satisfies a constraint described by a block object.

```
- (NSSet *)keysOfEntriesWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL
    (^)(id key, id obj, BOOL *stop))predicate
```

Parameters

opts

A bit mask of enumeration options.

predicate

A block object that specifies constraints for values in the receiver.

Return Value

The set of keys whose corresponding value satisfies *predicate*.

Availability

Available in iOS 4.0 and later.

See Also

- enumerateKeysAndObjectsWithOptions:usingBlock: (page 398)

Declared In

NSDictionary.h

keysSortedByValueUsingComparator:

Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using a given comparator block.

```
- (NSArray *)keysSortedByValueUsingComparator:(NSComparator)cmptr
```

Parameters

cmptr

A comparator block.

Return Value
An array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using *cmptr*.

Availability
Available in iOS 4.0 and later.

See Also
- keysSortedByValueWithOptions:usingComparator: (page 412)

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 iOS 2.0 and later.

See Also
- allKeys (page 394)
- sortedArrayUsingSelector: (page 80) (NSArray)

Declared In
NSDictionary.h

keysSortedByValueWithOptions:usingComparator:

Returns an array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using a given comparator block and a specified set of options.

```
- (NSArray *)keysSortedByValueWithOptions:(NSSortOptions)opts
    usingComparator:(NSComparator)cmptr
```

Parameters

opts

A bitmask of sort options.

cmptr

A comparator block.

Return Value

An array of the receiver's keys, in the order they would be in if the receiver were sorted by its values using *cmptr* with the options given in *opts*.

Availability

Available in iOS 4.0 and later.

See Also

- `keysSortedByValueUsingComparator:` (page 411)

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 395) method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

Availability

Available in iOS 2.0 and later.

See Also

- `keyEnumerator` (page 410)
- `nextObject` (page 424) (`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 iOS 2.0 and later.

See Also

- `allKeys` (page 394)
- `allValues` (page 395)
- `getObjects:andKeys:` (page 405)

Related Sample Code

BonjourWeb
CryptoExercise
KeyboardAccessory
MoviePlayer

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 iOS 2.0 and later.

See Also

- `allKeys` (page 394)
- `allValues` (page 395)

- getObjects:andKeys: (page 405)

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 414). If *key* does start with "@", strips the "@" and invokes [super valueForKey:] with the rest of the key.

Availability
Available in iOS 2.0 and later.

See Also
- setValue:forKey: (page 781) (NSMutableDictionary)
- getObjects:andKeys: (page 405)

Related Sample Code
GKTank

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 1267) 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 390) or the instance method initWithContentsOfFile: (page 405).

For more information about property lists, see *Property List Programming Guide*.

Availability
Available in iOS 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 390) or the instance method initWithContentsOfURL: (page 406).

For more information about property lists, see *Property List Programming Guide*.

Availability
Available in iOS 2.0 and later.

Declared In
`NSDictionary.h`

NSDirectoryEnumerator Class Reference

Inherits from	NSEnumerator : NSObject
Conforms to	NSFastEnumeration (NSEnumerator) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSFileManager.h
Companion guide	Low-Level File Management Programming Topics

Overview

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.

You obtain a directory enumerator using `NSFileManager`'s `enumeratorAtPath:` (page 512) method. For more details, see *Low-Level File Management Programming Topics*.

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 420)
  Returns an `NSDictionary` object that contains the attributes of the directory at which enumeration started.
- `fileAttributes` (page 420)
  Returns an object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).
- `level` (page 421)
  Returns the number of levels deep the current object is in the directory hierarchy being enumerated.

Skipping Subdirectories

- `skipDescendents` (page 421)
  Causes the receiver to skip recursion into the most recently obtained subdirectory.
- `skipDescendants` (page 421)
  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 515) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

Availability
Available in iOS 2.0 and later.

See Also
`createDirectoryAtPath:attributes:` (page 505) (`NSFileManager`)

Declared In
`NSFileManager.h`

fileAttributes

Returns an object that contains the attributes of the most recently returned file or subdirectory (as referenced by the pathname).

```
- (NSDictionary *)fileAttributes
```

Return Value
A dictionary 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 515) method of `NSFileManager` for details on obtaining the attributes from the dictionary.

Availability
Available in iOS 2.0 and later.

Declared In
NSFileManager.h

level

Returns the number of levels deep the current object is in the directory hierarchy being enumerated.

```
- (NSUInteger)level
```

Return Value
The number of levels, with the directory passed to `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` (page 513) (`NSFileManager`) considered to be level `0`.

Availability
Available in iOS 4.0 and later.

Declared In
NSFileManager.h

skipDescendants

Causes the receiver to skip recursion into the most recently obtained subdirectory.

```
- (void)skipDescendants
```

Discussion
This method is identical to `skipDescendents` (page 421) except for the spelling.

Availability
Available in iOS 4.0 and later.

Declared In
NSFileManager.h

skipDescendents

Causes the receiver to skip recursion into the most recently obtained subdirectory.

```
- (void)skipDescendents
```

Availability
Available in iOS 2.0 and later.

See Also
– `skipDescendants` (page 421)

Declared In
NSFileManager.h

NSEnumerator Class Reference

Inherits from	NSObject
Conforms to	NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSEnumerator.h
Companion guide	Collections Programming Topics

Overview

`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 1152) and `reverseObjectEnumerator` (page 76). `NSDictionary` also has two methods that return an `NSEnumerator` object: `keyEnumerator` (page 410) and `objectEnumerator` (page 413). These methods let you enumerate the contents of a dictionary by key or by value, respectively.

You send `nextObject` (page 424) 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 424)
  Returns an array of objects the receiver has yet to enumerate.
- `nextObject` (page 424)
  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 424) messages.

Invoking this method exhausts the enumerator's collection so that subsequent invocations of `nextObject` return `nil`.

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSError.h Foundation/NSURLError.h
Companion guide	Error Handling Programming Guide
Related sample code	AddMusic CryptoExercise GKRocket GKTank WiTap

Overview

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. Foundation error codes are found in the Cocoa error domain and documented in the *Foundation Constants Reference*. 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 431).

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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

Tasks

Creating Error Objects

`+ errorWithDomain:code:userInfo:` (page 429)
Creates and initializes an `NSError` object for a given domain and code with a given `userInfo` dictionary.

`- initWithDomain:code:userInfo:` (page 431)
Returns an `NSError` object initialized for a given domain and code with a given `userInfo` dictionary.

Getting Error Properties

`- code` (page 429)
Returns the receiver's error code.

`- domain` (page 430)
Returns the receiver's error domain.

`- userInfo` (page 434)
Returns the receiver's user info dictionary.

Getting a Localized Error Description

`- localizedDescription` (page 431)
Returns a string containing the localized description of the error.

`- localizedRecoveryOptions` (page 432)
Returns an array containing the localized titles of buttons appropriate for displaying in an alert panel.

`- localizedRecoverySuggestion` (page 433)
Returns a string containing the localized recovery suggestion for the error.

`- localizedFailureReason` (page 432)
Returns a string containing the localized explanation of the reason for the error.

Getting the Error Recovery Attempter

- `recoveryAttempter` (page 433)
  Returns an object that conforms to the `NSErrorRecoveryAttempting` informal protocol.

Displaying a Help Anchor

- `helpAnchor` (page 430)
  A string to display in response to an alert panel help anchor button being pressed.

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

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

See Also
- `localizedDescription` (page 431)
- `domain` (page 430)
- `userInfo` (page 434)

Declared In
`NSError.h`

domain

Returns the receiver's error domain.

```
- (NSString *)domain
```

Return Value
A string containing the receiver's error domain.

Availability

See Also
- `code` (page 429)
- `localizedDescription` (page 431)
- `userInfo` (page 434)

Declared In
`NSError.h`

helpAnchor

A string to display in response to an alert panel help anchor button being pressed.

```
- (NSString *)helpAnchor
```

Return Value
An NSString that the alert panel will include a help anchor button with that value.

Discussion
If this method returns a non-`nil` value for an error being presented by `alertWithError:`, the alert panel will include a help anchor button that can display this string.

The best way to get a value to return for this method is to specify it as the value of `NSHelpAnchorErrorKey` (page 436) in the `NSError` object's `userInfo` dictionary; or the method can be overridden.

Availability
Available in iOS 4.0 and later.

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

See Also
+ errorWithDomain:code:userInfo: (page 429)

Related Sample Code
CryptoExercise
WiTap

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

See Also
- `code` (page 429)
- `domain` (page 430)
- `userInfo` (page 434)

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 iOS 2.0 and later.

See Also
- `code` (page 429)
- `domain` (page 430)
- `userInfo` (page 434)

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 433). 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 iOS 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 iOS 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 432).

Availability
Available in iOS 2.0 and later.

See Also
`- localizedRecoveryOptions` (page 432)

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

See Also
- `code` (page 429)
- `domain` (page 430)
- `localizedDescription` (page 431)

Declared In
`NSError.h`

Constants

User info dictionary keys

These keys may exist in the user info dictionary.

```
NSString * const NSLocalizedDescriptionKey;
NSString * const NSErrorFailingURLStringKey;
NSString * const NSFilePathErrorKey;
NSString * const NSStringEncodingErrorKey;
NSString * const NSUnderlyingErrorKey;
NSString * const NSURLErrorKey;
NSString * const NSLocalizedFailureReasonErrorKey;
NSString * const NSLocalizedRecoverySuggestionErrorKey;
NSString * const NSLocalizedRecoveryOptionsErrorKey;
NSString * const NSRecoveryAttempterErrorKey;
NSString * const NSHelpAnchorErrorKey;
NSString * const NSURLErrorFailingURLErrorKey;
NSString * const NSURLErrorFailingURLStringErrorKey;
NSString * const NSURLErrorFailingURLPeerTrustErrorKey;
```

Constants
`NSLocalizedDescriptionKey`
The corresponding value is a localized string representation of the error that, if present, will be returned by `localizedDescription` (page 431).

Available in iOS 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`. (Deprecated. This constant is deprecated in Mac OS X 10.6, and is superseded by `NSURLErrorFailingURLStringErrorKey` (page 436).)

This constant is deprecated in Mac OS X 10.6, and is superseded by `NSURLErrorFailingURLStringErrorKey` (page 436). Both constants refer to the same value for backward-compatibility, but the new symbol name has a better prefix.

Available in iOS 2.0 and later.

Deprecated in iOS 4.0.

Declared in `NSURLError.h`.

`NSFilePathErrorKey`

Contains the file path of the error.

The corresponding value is an `NSString` object.

Available in iOS 2.0 and later.

Declared in `NSError.h`.

`NSStringEncodingErrorKey`

The corresponding value is an `NSNumber` object containing the `NSStringEncoding` value.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSError.h`.

`NSURLErrorKey`

The corresponding value is an `NSURL` object.

Available in iOS 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 432).

This string provides a more detailed explanation of the error than the description.

Available in iOS 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 iOS 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 433).

Available in iOS 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 433).

Available in iOS 2.0 and later.

Declared in `NSError.h`.

`NSHelpAnchorErrorKey`

The corresponding value is an `NSString` containing the localized help corresponding to the help button. See `helpAnchor` (page 430) for more information.

Available in iOS 4.0 and later.

Declared in `NSError.h`.

`NSURLErrorFailingURLErrorKey`

The corresponding value is an `NSURL` containing the URL which caused a load to fail. This key is only present in the `NSURLErrorDomain`.

Available in iOS 4.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorFailingURLStringErrorKey`

The corresponding value is an `NSString` object for the URL which caused a load to fail. This key is only present in the `NSURLErrorDomain`.

This constant supersedes `NSErrorFailingURLStringKey` (page 435), which was deprecated in Mac OS X 10.6. Both constants refer to the same value for backward-compatibility, but this symbol name has a better prefix.

Available in iOS 4.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorFailingURLPeerTrustErrorKey`

The corresponding value is the `SecTrustRef` object representing the state of a failed SSL handshake. This key is only present in the `NSURLErrorDomain`.

Available in iOS 3.0 and later.

Declared in `NSURLError.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 iOS 2.0 and later.

Declared in `NSError.h`.

`NSOSStatusErrorDomain`

Mac OS 9/Carbon errors

Available in iOS 2.0 and later.

Declared in `NSError.h`.

`NSMachErrorDomain`

Mach errors

Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSException.h
Companion guide	Exception Programming Topics

Overview

`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 key-press (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 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Creating and Raising an NSException Object

`+ exceptionWithName:reason:userInfo:` (page 440)
Creates and returns an exception object .

`+ raise:format:` (page 441)
A convenience method that creates and raises an exception.

`+ raise:format:arguments:` (page 442)
Creates and raises an exception with the specified name, reason, and arguments.

`- initWithName:reason:userInfo:` (page 443)
Initializes and returns a newly allocated exception object.

`- raise` (page 444)
Raises the receiver, causing program flow to jump to the local exception handler.

Querying an NSException Object

`- name` (page 443)
Returns an `NSString` object used to uniquely identify the receiver.

`- reason` (page 444)
Returns an `NSString` object containing a “human-readable” reason for the receiver.

`- userInfo` (page 445)
Returns an `NSDictionary` object containing application-specific data pertaining to the receiver.

Getting Exception Stack Frames

`- callStackReturnAddresses` (page 442)
Returns the call return addresses related to a raised exception.

`- callStackSymbols` (page 443)
Returns an array containing the current call symbols.

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 iOS 2.0 and later.

See Also

- `initWithName:reason:userInfo:` (page 443)
- `name` (page 443)
- `reason` (page 444)
- `userInfo` (page 445)

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 iOS 2.0 and later.

See Also

- `raise:format:arguments:` (page 442)
- `raise` (page 444)

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 iOS 2.0 and later.

See Also
`+ raise:format:` (page 441)
`- raise` (page 444)

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` (page 1753) 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 iOS 2.0 and later.

Declared In
`NSException.h`

callStackSymbols

Returns an array containing the current call symbols.

```
- (NSArray *)callStackSymbols
```

Return Value
An array containing the current call stack symbols.

Discussion
This method returns an array of strings describing the call stack backtrace at the moment the exception was first raised. The format of each string is non-negotiable and is determined by the `backtrace_symbols()` API

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 440)

Declared In
`NSException.h`

name

Returns an `NSString` object used to uniquely identify the receiver.

```
- (NSString *)name
```

Availability
Available in iOS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 440)
`- initWithName:reason:userInfo:` (page 443)

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 iOS 2.0 and later.

See Also
`+ raise:format:` (page 441)
`+ raise:format:arguments:` (page 442)

Declared In
`NSException.h`

reason

Returns an `NSString` object containing a “human-readable” reason for the receiver.

```
- (NSString *)reason
```

Availability
Available in iOS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 440)
`- initWithName:reason:userInfo:` (page 443)

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 iOS 2.0 and later.

See Also
`+ exceptionWithName:reason:userInfo:` (page 440)
`- initWithName:reason:userInfo:` (page 443)

Declared In
`NSException.h`

NSExpression Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	Foundation/NSExpression.h
Companion guide	Predicate Programming Guide

Overview

`NSExpression` is used to represent expressions in a predicate.

Comparison operations in an `NSPredicate` are based on two expressions, as represented by instances of the `NSExpression` class. Expressions are created for constant values, key paths, and so on.

Generally, anywhere in the `NSExpression` class hierarchy where there is composite API and subtypes that may only reasonably respond to a subset of that API, invoking a method that does not make sense for that subtype will cause an exception to be thrown.

Expression Types

In Mac OS X v10.5, `NSExpression` introduces several new expression types: `NSSubqueryExpressionType`, `NSAggregateExpressionType`, `NSUnionSetExpressionType`, `NSIntersectSetExpressionType`, and `NSMinusSetExpressionType`.

Aggregate Expressions

The aggregate expression allows you to create predicates containing expressions that evaluate to collections that contain further expressions. The collection may be an `NSArray`, `NSSet`, or `NSDictionary` object.

For example, consider the BETWEEN operator (`NSBetweenPredicateOperatorType` (page 229)); its right hand side is a collection containing two elements. Using just the Mac OS X v10.4 API, these elements must be constants, as there is no way to populate them using variable expressions. On Mac OS X v10.4, it is not possible to create a predicate template to the effect of `date between {$YESTERDAY, $TOMORROW}`; instead you must create a new predicate each time.

Aggregate expressions are not supported by Core Data.

Subquery Expressions

The `NSSubqueryExpressionType` (page 467) creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

Set Expressions

The set expressions (`NSUnionSetExpressionType` (page 467), `NSIntersectSetExpressionType` (page 467), and `NSMinusSetExpressionType` (page 467)) combine results in a manner similar to the `NSSet` methods.

Both sides of these expressions must evaluate to a collection; the left-hand side must evaluate to an `NSSet` object, the right-hand side can be any other collection type.

```
(expression UNION expression)
(expression INTERSECT expression)
(expression MINUS expression)
```

Set expressions are not supported by Core Data.

Function Expressions

On Mac OS X v10.4, `NSExpression` only supports a predefined set of functions: `sum`, `count`, `min`, `max`, and `average`. These predefined functions were accessed in the predicate syntax using custom keywords (for example, `MAX(1, 5, 10)`).

On Mac OS X v10.5 and later, function expressions also support arbitrary method invocations. To use this extended functionality, you can now use the syntax `FUNCTION(receiver, selectorName, arguments, ...)`, for example:

```
FUNCTION(@"/Developer/Tools/otest", @"lastPathComponent") => @"otest"
```

All methods must take 0 or more `id` arguments and return an `id` value, although you can use the `CAST` expression to convert datatypes with lossy string representations (for example, `CAST(####, "NSDate")`). The `CAST` expression is extended in Mac OS X v10.5 to provide support for casting to classes for use in creating receivers for function expressions.

Note that although Core Data supports evaluation of the predefined functions, it does not support the evaluation of custom predicate functions in the persistent stores (during a fetch).

Tasks

Initializing an Expression

`- initWithExpressionType:` (page 463)
Initializes the receiver with the specified expression type.

Creating an Expression for a Value

`+ expressionForConstantValue:` (page 452)
Returns a new expression that represents a given constant value.

`+ expressionForEvaluatedObject` (page 452)
Returns a new expression that represents the object being evaluated.

`+ expressionForKeyPath:` (page 458)
Returns a new expression that invokes `valueForKeyPath:` with a given key path.

`+ expressionForVariable:` (page 461)
Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

Creating a Collection Expression

`+ expressionForAggregate:` (page 451)
Returns a new aggregate expression for a given collection.

`+ expressionForUnionSet:with:` (page 460)
Returns a new `NSExpression` object that represent the union of a given set and collection.

`+ expressionForIntersectSet:with:` (page 458)
Returns a new `NSExpression` object that represent the intersection of a given set and collection.

`+ expressionForMinusSet:with:` (page 458)
Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

Creating a Subquery

`+ expressionForSubquery:usingIteratorVariable:predicate:` (page 459)
Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

Creating an Expression Using Blocks

`+ expressionForBlock:arguments:` (page 451)
Creates an NSExpression object that will use the Block for evaluating objects.

Creating an Expression for a Function

+ `expressionForFunction:arguments:` (page 453)
  Returns a new expression that will invoke one of the predefined functions.

+ `expressionForFunction:selectorName:arguments:` (page 457)
  Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

Getting Information About an Expression

- `arguments` (page 461)
  Returns the arguments for the receiver.

- `collection` (page 461)
  Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

- `constantValue` (page 462)
  Returns the constant value of the receiver.

- `expressionType` (page 462)
  Returns the expression type for the receiver.

- `function` (page 463)
  Returns the function for the receiver.

- `keyPath` (page 464)
  Returns the key path for the receiver.

- `leftExpression` (page 464)
  Returns the left expression of an aggregate expression.

- `operand` (page 465)
  Returns the operand for the receiver.

- `predicate` (page 465)
  Return the predicate of a subquery expression.

- `rightExpression` (page 465)
  Returns the right expression of an aggregate expression.

- `variable` (page 466)
  Returns the variable for the receiver.

Evaluating an Expression

- `expressionValueWithObject:context:` (page 463)
  Evaluates an expression using a given object and context.

Accessing the Expression Block

- `expressionBlock` (page 462)
  Returns the expression's expression Block.

Class Methods

expressionForAggregate:

Returns a new aggregate expression for a given collection.

```
+ (NSExpression *)expressionForAggregate:(NSArray *)collection
```

Parameters

collection

A collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`) that contains further expressions.

Return Value

A new expression that contains the expressions in *collection*.

Availability

Available in iOS 3.0 and later.

Declared In

`NSExpression.h`

expressionForBlock:arguments:

Creates an NSExpression object that will use the Block for evaluating objects.

```
+ (NSExpression *)expressionForBlock:(id (^)(id evaluatedObject, NSArray
    *expressions, NSMutableDictionary *context))blockarguments:(NSArray *)arguments
```

Parameters

block

The Block is applied to the object to be evaluated.

The Block takes three arguments and returns a value:

evaluatedObject

The object to be evaluated.

expressions

An array of predicate expressions that evaluates to a collection.

context

A dictionary that the expression can use to store temporary state for one predicate evaluation.

Note that *context* is mutable, and that it can only be accessed during the evaluation of the expression. You must not attempt to retain it for use elsewhere.]

The Block returns the evaluatedObject.

arguments

An array containing `NSExpression` objects that will be used as parameters during the invocation of selector.

For a selector taking no parameters, the array should be empty. For a selector taking one or more parameters, the array should contain one `NSExpression` object which will evaluate to an instance of the appropriate type for each parameter.

If there is a mismatch between the number of parameters expected and the number you provide during evaluation, an exception may be raised or missing parameters may simply be replaced by `nil` (which occurs depends on how many parameters are provided, and whether you have over- or underflow).

See `expressionForFunction:arguments:` (page 453) for a complete list of arguments.

Return Value
An expression that filters a collection using the specified Block.

Availability
Available in iOS 4.0 and later.

See Also
`- expressionBlock` (page 462)

Declared In
`NSExpression.h`

expressionForConstantValue:

Returns a new expression that represents a given constant value.

```
+ (NSExpression *)expressionForConstantValue:(id)obj
```

Parameters

obj

The constant value the new expression is to represent.

Return Value
A new expression that represents the constant value, *obj*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionForEvaluatedObject

Returns a new expression that represents the object being evaluated.

```
+ (NSExpression *)expressionForEvaluatedObject
```

Return Value
A new expression that represents the object being evaluated.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

expressionForFunction:arguments:

Returns a new expression that will invoke one of the predefined functions.

```
+ (NSExpression *)expressionForFunction:(NSString *)name arguments:(NSArray
    *)parameters
```

Parameters

name

The name of the function to invoke.

parameters

An array containing NSExpression objects that will be used as parameters during the invocation of selector.

For a selector taking no parameters, the array should be empty. For a selector taking one or more parameters, the array should contain one NSExpression object which will evaluate to an instance of the appropriate type for each parameter.

If there is a mismatch between the number of parameters expected and the number you provide during evaluation, an exception may be raised or missing parameters may simply be replaced by nil (which occurs depends on how many parameters are provided, and whether you have over- or underflow).

Return Value
A new expression that invokes the function *name* using the parameters in *parameters*.

Discussion
The *name* parameter can be one of the following predefined functions.

Function	Parameter	Returns	Availability
average:	An NSArray object containing NSExpression objects representing numbers	An NSNumber object (the average of values in the array)	Mac OS X v10.4 and later
sum:	An NSArray object containing NSExpression objects representing numbers	An NSNumber object (the sum of values in the array)	Mac OS X v10.4 and later
count:	An NSArray object containing NSExpression objects representing numbers	An NSNumber object (the number of elements in the array)	Mac OS X v10.4 and later
min:	An NSArray object containing NSExpression objects representing numbers	An NSNumber object (the minimum of the values in the array)	Mac OS X v10.4 and later

Function	Parameter	Returns	Availability
`max:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the maximum of the values in the array)	Mac OS X v10.4 and later
`median:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the median of the values in the array)	Mac OS X v10.5 and later
`mode:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSArray` object (the mode of the values in the array)	Mac OS X v10.5 and later
`stddev:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the standard deviation of the values in the array)	Mac OS X v10.5 and later
`add:to:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the sum of the values in the array)	Mac OS X v10.5 and later
`from:subtract:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of subtracting the second value in the array from the first value in the array)	Mac OS X v10.5 and later
`multiply:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of multiplying the values in the array)	Mac OS X v10.5 and later
`divide:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`modulus:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the remainder of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`sqrt:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the square root of the value in the array)	Mac OS X v10.5 and later
`log:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the log of the value in the array)	Mac OS X v10.5 and later
`ln:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the natural log of the value in the array)	Mac OS X v10.5 and later

Function	Parameter	Returns	Availability
`raise:toPower:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of raising the first value in the array to the power of the second value in the array)	Mac OS X v10.5 and later
`exp:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the base-e exponential of the value in the array)	Mac OS X v10.5 and later
`ceiling:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the smallest integral value not less than the value in the array)	Mac OS X v10.5 and later
`abs:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the absolute value of the value in the array)	Mac OS X v10.5 and later
`trunc:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the integral value nearest to but no greater than the value in the array)	Mac OS X v10.5 and later
`random`	`nil`	An `NSNumber` object (a random integer value)	Mac OS X v10.5 and later
`random:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (a random integer value between `0` and the value in the array (exclusive))	Mac OS X v10.5 and later
`now`	`nil`	An `[NSDate]` object (the current date and time)	Mac OS X v10.5 and later
`floor:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object	iOS 3.0 and later
`uppercase:`	An `NSArray` object containing one `NSExpression` object representing a string	An `NSString` object	iOS 3.0 and later
`lowercase:`	An `NSArray` object containing one `NSExpression` object representing a string	An `NSString` object	iOS 3.0 and later
`bitwiseAnd:with:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later
`bitwiseOr:with:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later

Function	Parameter	Returns	Availability
`bitwiseXor:with:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later
`leftshift:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later
`rightshift:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later
`onesComplement:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the number is treated as an `NSInteger`)	iOS 3.0 and later
`noindex:`	An `NSArray` object containing an `NSExpression` object	The result of evaluating the parameter as though the `noindex:` function expression didn't exist.	iOS 3.0 and later

This method raises an exception immediately if the selector is invalid; it raises an exception at runtime if the parameters are incorrect.

The *`parameters`* argument is a collection containing an expression which evaluates to a collection, as illustrated in the following examples:

```
NSNumber *number1 = [NSNumber numberWithInteger:20];
NSNumber *number2 = [NSNumber numberWithInteger:40];
NSArray *numberArray = [NSArray arrayWithObjects: number1, number2, nil];

NSExpression *arrayExpression = [NSExpression expressionForConstantValue:
numberArray];
NSArray *argumentArray = [NSArray arrayWithObject: arrayExpression];

NSExpression* expression =
    [NSExpression expressionForFunction:@"sum:" arguments:argumentArray];
id result = [expression expressionValueWithObject: nil context: nil];

BOOL ok = [result isEqual: [NSNumber numberWithInt: 60]]; // ok == YES

[NSExpression expressionForFunction:@"random" arguments:nil];

[NSExpression expressionForFunction:@"max:"
    arguments: [NSArray arrayWithObject:
        [NSExpression expressionForConstantValue:
            [NSArray arrayWithObjects:
                [NSNumber numberWithInt: 5], [NSNumber numberWithInt: 10],
nil]]]];

[NSExpression expressionForFunction:@"subtract:from:"
    arguments: [NSArray arrayWithObjects:
        [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 5]],
```

```
      [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 10]],
 nil]];
```

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

Availability
Available in iOS 3.0 and later.

See Also
`+ expressionForFunction:selectorName:arguments:` (page 457)

Declared In
`NSExpression.h`

expressionForFunction:selectorName:arguments:

Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

```
+ (NSExpression *)expressionForFunction:(NSExpression *)target selectorName:(NSString
     *)name arguments:(NSArray *)parameters
```

Parameters

target
: An `NSExpression` object which will evaluate an object on which the selector identified by *name* may be invoked.

name
: The name of the method to be invoked.

parameters
: An array containing `NSExpression` objects which can be evaluated to provide parameters for the method specified by *name*.

Return Value
An expression which will return the result of invoking the selector named *name* on the result of evaluating the target expression with the parameters specified by evaluating the elements of *parameters*.

Discussion
See the description of `expressionForFunction:arguments:` (page 453) for examples of how to construct the parameter array.

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

This expression effectively allows your application to invoke any method on any object it can navigate to at runtime. You must consider the security implications of this type of evaluation.

Availability
Available in iOS 3.0 and later.

See Also
`+ expressionForFunction:arguments:` (page 453)

Declared In
`NSExpression.h`

expressionForIntersectSet:with:

Returns a new `NSExpression` object that represent the intersection of a given set and collection.

```
+ (NSExpression *)expressionForIntersectSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left

An expression that evaluates to an `NSSet` object.

right

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value
A new `NSExpression` object that represents the intersection of *left* and *right*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionForKeyPath:

Returns a new expression that invokes `valueForKeyPath:` with a given key path.

```
+ (NSExpression *)expressionForKeyPath:(NSString *)keyPath
```

Parameters

keyPath

The key path that the new expression should evaluate.

Return Value
A new expression that invokes `valueForKeyPath:` (page 1590) with *keyPath*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionForMinusSet:with:

Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

```
+ (NSExpression *)expressionForMinusSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left

An expression that evaluates to an `NSSet` object.

right

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value

A new `NSExpression` object that represents the subtraction of *right* from *left*.

Availability

Available in iOS 3.0 and later.

Declared In

`NSExpression.h`

expressionForSubquery:usingIteratorVariable:predicate:

Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

```
+ (NSExpression *)expressionForSubquery:(NSExpression *)expression
    usingIteratorVariable:(NSString *)variable predicate:(id)predicate
```

Parameters

expression

A predicate expression that evaluates to a collection.

variable

Used as a local variable, and will shadow any instances of variable in the bindings dictionary. The variable is removed or the old value replaced once evaluation completes.

predicate

The predicate used to determine whether the element belongs in the result collection.

Return Value

An expression that filters a collection by storing elements in the collection in the variable variable and keeping the elements for which qualifier returns true

Discussion

This method creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

For example, suppose you have an Apartment entity that has a to-many relationship to a Resident entity, and that you want to create a query for all apartments inhabited by a resident whose first name is "Jane" and whose last name is "Doe". Using only API available for Mac OS X v 10.4, you could try the predicate:

```
resident.firstname == "Jane" && resident.lastname == "Doe"
```

but this will always return false since `resident.firstname` and `resident.lastname` both return collections. You could also try:

```
resident.firstname CONTAINS "Jane" && resident.lastname CONTAINS "Doe"
```

but this is also flawed—it returns true if there are two residents, one of whom is John Doe and one of whom is Jane Smith. The only way to find the desired apartments is to do two passes: one through residents to find "Jane Doe", and one through apartments to find the ones where our Jane Does reside.

Subquery expressions provide a way to encapsulate this type of qualification into a single query.

The string format for a subquery expression is:

```
SUBQUERY(collection_expression, variable_expression, predicate);
```

where `expression` is a predicate expression that evaluates to a collection, `variableExpression` is an expression which will be used to contain each individual element of `collection`, and `predicate` is the predicate used to determine whether the element belongs in the result collection.

Using subqueries, the apartment query could be reformulated as

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe").@count
 != 0)
```

or

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe")[size]
 != 0)
```

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionForUnionSet:with:

Returns a new `NSExpression` object that represent the union of a given set and collection.

```
+ (NSExpression *)expressionForUnionSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

`left`

An expression that evaluates to an `NSSet` object.

`right`

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value
An new `NSExpression` object that represents the union of *`left`* and *`right`*.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionForVariable:

Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

```
+ (NSExpression *)expressionForVariable:(NSString *)string
```

Parameters

string

The key for the variable to extract from the variable bindings dictionary.

Return Value

A new expression that extracts from the variable bindings dictionary the value for the key *string*.

Availability

Available in iOS 3.0 and later.

Declared In

NSExpression.h

Instance Methods

arguments

Returns the arguments for the receiver.

```
- (NSArray *)arguments
```

Return Value

The arguments for the receiver—that is, the array of expressions that will be passed as parameters during invocation of the selector on the operand of a function expression.

Discussion

This method raises an exception if it is not applicable to the receiver.

Availability

Available in iOS 3.0 and later.

Declared In

NSExpression.h

collection

Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

```
- (id)collection
```

Return Value

Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

constantValue

Returns the constant value of the receiver.

```
- (id)constantValue
```

Return Value
The constant value of the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

expressionBlock

Returns the expression's expression Block.

```
- (id, NSArray *, NSMutableDictionary *)expressionBlock
```

Return Value
The expression's expression Block as created in `expressionForBlock:arguments:` (page 451).

Availability
Available in iOS 4.0 and later.

See Also
`+ expressionForBlock:arguments:` (page 451)

Declared In
NSExpression.h

expressionType

Returns the expression type for the receiver.

```
- (NSExpressionType)expressionType
```

Return Value
The expression type for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

expressionValueWithObject:context:

Evaluates an expression using a given object and context.

```
- (id)expressionValueWithObject:(id)object context:(NSMutableDictionary *)context
```

Parameters

object

The object against which the receiver is evaluated.

context

A dictionary that the expression can use to store temporary state for one predicate evaluation. Can be `nil`.

Note that *context* is mutable, and that it can only be accessed during the evaluation of the expression. You must not attempt to retain it for use elsewhere.]

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

function

Returns the function for the receiver.

```
- (NSString *)function
```

Return Value
The function for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

initWithExpressionType:

Initializes the receiver with the specified expression type.

```
- (id)initWithExpressionType:(NSExpressionType)type
```

Parameters

type

The type of the new expression, as defined by `NSExpressionType` (page 466).

Return Value

An initialized `NSExpression` object of the type *type*.

Special Considerations

This method is the designated initializer for `NSExpression`.

Availability

Available in iOS 3.0 and later.

Declared In

`NSExpression.h`

keyPath

Returns the key path for the receiver.

```
- (NSString *)keyPath
```

Return Value

The key path for the receiver.

Discussion

This method raises an exception if it is not applicable to the receiver.

Availability

Available in iOS 3.0 and later.

Declared In

`NSExpression.h`

leftExpression

Returns the left expression of an aggregate expression.

```
- (NSExpression *)leftExpression
```

Return Value

The left expression of a set expression.

Discussion

This method raises an exception if it is not applicable to the receiver.

Availability

Available in iOS 3.0 and later.

Declared In

`NSExpression.h`

operand

Returns the operand for the receiver.

```
- (NSExpression *)operand
```

Return Value
The operand for the receiver—that is, the object on which the selector will be invoked.

Discussion
The object is the result of evaluating a key path or one of the defined functions. This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

predicate

Return the predicate of a subquery expression.

```
- (NSPredicate *)predicate
```

Return Value
The predicate of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

rightExpression

Returns the right expression of an aggregate expression.

```
- (NSExpression *)rightExpression
```

Return Value
The right expression of a set expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
NSExpression.h

variable

Returns the variable for the receiver.

```
- (NSString *)variable
```

Return Value
The variable for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

Constants

NSExpressionType

Defines the possible types of `NSExpression`.

```
enum {
    NSConstantValueExpressionType = 0,
    NSEvaluatedObjectExpressionType,
    NSVariableExpressionType,
    NSKeyPathExpressionType,
    NSFunctionExpressionType,
    NSAggregateExpressionType,
    NSSubqueryExpressionType = 13,
    NSUnionSetExpressionType,
    NSIntersectSetExpressionType,
    NSMinusSetExpressionType,
    NSBlockExpressionType = 19
}
typedef NSUInteger NSExpressionType;
```

Constants

`NSConstantValueExpressionType`
An expression that always returns the same value.
Available in iOS 3.0 and later.
Declared in `NSExpression.h`.

`NSEvaluatedObjectExpressionType`
An expression that always returns the parameter object itself.
Available in iOS 3.0 and later.
Declared in `NSExpression.h`.

`NSVariableExpressionType`
An expression that always returns whatever value is associated with the key specified by 'variable' in the bindings dictionary.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSKeyPathExpressionType`
An expression that returns something that can be used as a key path.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSFunctionExpressionType`
An expression that returns the result of evaluating a function.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSAggregateExpressionType`
An expression that defines an aggregate of `NSExpression` objects.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSSubqueryExpressionType`
An expression that filters a collection using a subpredicate.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSUnionSetExpressionType`
An expression that creates a union of the results of two nested expressions.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSIntersectSetExpressionType`
An expression that creates an intersection of the results of two nested expressions.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSMinusSetExpressionType`
An expression that combines two nested expression results by set subtraction.

Available in iOS 3.0 and later.

Declared in `NSExpression.h`.

`NSBlockExpressionType`
An expression that uses an Block.

Available in iOS 4.0 and later.

Declared in `NSExpression.h`.

Availability
Available in iOS 3.0 and later.

Declared In
`NSExpression.h`

NSFileHandle Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSFileHandle.h
Companion guide	Low-Level File Management Programming Topics

Overview

`NSFileHandle` objects provide an object-oriented wrapper for accessing open files or communications channels.

See the *PictureSharing* example project to examine code that creates an NSFileHandle object to listen for incoming connections; the file-handle object is initialized from a socket obtained through BSD calls.

Note: The deallocation of an `NSFileHandle` object deletes its descriptor and closes the represented file or channel unless the `NSFileHandle` object was created with `initWithFileDescriptor:` (page 479) or `initWithFileDescriptor:closeOnDealloc:` (page 480) with `NO` as the parameter argument.

Tasks

Getting a File Handle

`+ fileHandleForReadingAtPath:` (page 472)
Returns a file handle initialized for reading the file, device, or named socket at the specified path.

`+ fileHandleForReadingFromURL:error:` (page 472)
Returns a file handle initialized for reading the file, device, or named socket at the specified URL.

`+ fileHandleForWritingAtPath:` (page 474)
Returns a file handle initialized for writing to the file, device, or named socket at the specified path.

`+ fileHandleForWritingToURL:error:` (page 475)
Returns a file handle initialized for writing to the file, device, or named socket at the specified URL.

+ `fileHandleForUpdatingAtPath:` (page 473)
  Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified path.
+ `fileHandleForUpdatingURL:error:` (page 473)
  Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified URL.
+ `fileHandleWithStandardError` (page 476)
  Returns the file handle associated with the standard error file.
+ `fileHandleWithStandardInput` (page 476)
  Returns the file handle associated with the standard input file.
+ `fileHandleWithStandardOutput` (page 476)
  Returns the file handle associated with the standard output file.
+ `fileHandleWithNullDevice` (page 475)
  Returns a file handle associated with a null device.

Creating a File Handle

- `initWithFileDescriptor:` (page 479)
  Returns a file handle initialized with a file descriptor.
- `initWithFileDescriptor:closeOnDealloc:` (page 480)
  Returns a file handle initialized with a file handle, using a specified deallocation policy.

Getting a File Descriptor

- `fileDescriptor` (page 479)
  Returns the file descriptor associated with the receiver.

Reading from a File Handle

- `availableData` (page 478)
  Returns the data available through the receiver.
- `readDataToEndOfFile` (page 482)
  Returns the data available through the receiver up to the end of file or maximum number of bytes.
- `readDataOfLength:` (page 481)
  Reads data up to a specified number of bytes from the receiver.

Writing to a File Handle

- `writeData:` (page 487)
  Synchronously writes data to the file, device, pipe, or socket represented by the receiver.

Communicating Asynchronously

- `acceptConnectionInBackgroundAndNotify` (page 477)
  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 478)
  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 482)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readInBackgroundAndNotifyForModes:` (page 483)
  Reads from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotify` (page 483)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 484)
  Reads to the end of file from the file or communications channel in the background and posts a notification when finished.
- `waitForDataInBackgroundAndNotify` (page 486)
  Checks to see if data is available in a background thread.
- `waitForDataInBackgroundAndNotifyForModes:` (page 486)
  Checks to see if data is available in a background thread.

Seeking Within a File

- `offsetInFile` (page 481)
  Returns the position of the file pointer within the file represented by the receiver.
- `seekToEndOfFile` (page 484)
  Puts the file pointer at the end of the file referenced by the receiver and returns the new file offset.
- `seekToFileOffset:` (page 485)
  Moves the file pointer to the specified offset within the file represented by the receiver.

Operating on a File

- `closeFile` (page 479)
  Disallows further access to the represented file or communications channel and signals end of file on communications channels that permit writing.
- `synchronizeFile` (page 485)
  Causes all in-memory data and attributes of the file represented by the receiver to be written to permanent storage.
- `truncateFileAtOffset:` (page 485)
  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 iOS 2.0 and later.

See Also

- `availableData` (page 478)
- `initWithFileDescriptor:` (page 479)
- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)

Declared In

`NSFileHandle.h`

fileHandleForReadingFromURL:error:

Returns a file handle initialized for reading the file, device, or named socket at the specified URL.

```
+ (id)fileHandleForReadingFromURL:(NSURL *)url error:(NSError **)error
```

Parameters

url

The URL of the file, device, or named socket to access.

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

The initialized file handle, or `nil` if no file exists at *url*.

Discussion

The file pointer is set to the beginning of the file. The returned object responds only to `NSFileHandleread...` messages.

Availability
Available in iOS 4.0 and later.

See Also
- `availableData` (page 478)
- `initWithFileDescriptor:` (page 479)
- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)

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 487).

Availability
Available in iOS 2.0 and later.

See Also
- `availableData` (page 478)
- `initWithFileDescriptor:` (page 479)
- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)

Declared In
`NSFileHandle.h`

fileHandleForUpdatingURL:error:

Returns a file handle initialized for reading and writing to the file, device, or named socket at the specified URL.

```
+ (id)fileHandleForUpdatingURL:(NSURL *)url error:(NSError **)error
```

Parameters
url
The URL of the file, device, or named socket to access.

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
The initialized file handle, or `nil` if no file exists at *url*.

Discussion
The file pointer is set to the beginning of the file. The returned object responds to both `NSFileHandleread...` messages and `writeData:` (page 487).

Availability
Available in iOS 4.0 and later.

See Also
- `availableData` (page 478)
- `initWithFileDescriptor:` (page 479)
- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)
- `writeData:` (page 487)

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 487).

Availability
Available in iOS 2.0 and later.

See Also
- `initWithFileDescriptor:` (page 479)
- `writeData:` (page 487)

Declared In
`NSFileHandle.h`

fileHandleForWritingToURL:error:

Returns a file handle initialized for writing to the file, device, or named socket at the specified URL.

```
+ (id)fileHandleForWritingToURL:(NSURL *)url error:(NSError **)error
```

Parameters

url

The URL of the file, device, or named socket to access.

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

The initialized file handle, or `nil` if no file exists at *url*.

Discussion

The file pointer is set to the beginning of the file. The returned object responds only to `writeData:` (page 487).

Availability

Available in iOS 4.0 and later.

See Also

`- initWithFileDescriptor:` (page 479)

`- writeData:` (page 487)

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 479) returns an illegal value. Other methods are no-ops or return "sensible" values.

Availability

Available in iOS 2.0 and later.

See Also

`- initWithFileDescriptor:` (page 479)

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 iOS 2.0 and later.

See Also
`+ fileHandleWithNullDevice` (page 475)
`- initWithFileDescriptor:` (page 479)

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 iOS 2.0 and later.

See Also
`+ fileHandleWithNullDevice` (page 475)
`- initWithFileDescriptor:` (page 479)

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 iOS 2.0 and later.

See Also
`+ fileHandleWithNullDevice` (page 475)
`- initWithFileDescriptor:` (page 479)

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 489) 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 479) 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 489).

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 iOS 2.0 and later.

See Also
`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855) (`NSNotificationQueue`)
`- readInBackgroundAndNotify` (page 482)
`- readToEndOfFileInBackgroundAndNotify` (page 483)

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 477) for details of how this method operates. This method differs from `acceptConnectionInBackgroundAndNotify` (page 477) in that *modes* specifies the run-loop mode (or modes) in which `NSFileHandleConnectionAcceptedNotification` (page 489) can be posted.

Availability

Available in iOS 2.0 and later.

See Also

- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855) (`NSNotificationQueue`)
- `readInBackgroundAndNotifyForModes:` (page 483)
- `readToEndOfFileInBackgroundAndNotifyForModes:` (page 484)

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 iOS 2.0 and later.

See Also

- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)

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 represented by the receiver 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. The deallocation of an `NSFileHandle` object deletes its descriptor and closes the represented file or channel unless the `NSFileHandle` object was created with `initWithFileDescriptor:` (page 479) or `initWithFileDescriptor:closeOnDealloc:` (page 480) with `NO` as the parameter argument.

Availability
Available in iOS 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 479), this method raises an exception.

Availability
Available in iOS 2.0 and later.

See Also
`- initWithFileDescriptor:` (page 479)

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 iOS 2.0 and later.

See Also

- closeFile (page 479)

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 iOS 2.0 and later.

See Also

- closeFile (page 479)

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 iOS 2.0 and later.

See Also
- `seekToEndOfFile` (page 484)
- `seekToFileOffset:` (page 485)

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 iOS 2.0 and later.

See Also
- `availableData` (page 478)
- `readDataToEndOfFile` (page 482)

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 481) as part of its implementation.

Availability
Available in iOS 2.0 and later.

See Also
`- availableData` (page 478)

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 478) operation on a file or communications channel and posts an `NSFileHandleReadCompletionNotification` (page 489) 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 489). In communication via stream-type sockets, the receiver is often the object returned in the *`userInfo`* dictionary of `NSFileHandleConnectionAcceptedNotification` (page 489).

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 iOS 2.0 and later.

See Also
`- acceptConnectionInBackgroundAndNotify` (page 477)
`- readToEndOfFileInBackgroundAndNotifyForModes:` (page 484)
`- enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855) (`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 482) for details of how this method operates. This method differs from readInBackgroundAndNotify (page 482) in that *modes* specifies the run-loop mode (or modes) in which NSFileHandleReadCompletionNotification (page 489) can be posted.

Availability
Available in iOS 2.0 and later.

See Also
- acceptConnectionInBackgroundAndNotifyForModes: (page 478)
- enqueueNotification:postingStyle:coalesceMask:forModes: (page 855) (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 490) 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 490). In communication via stream-type sockets, the receiver is often the object returned in the *userInfo* dictionary of NSFileHandleConnectionAcceptedNotification (page 489).

Availability
Available in iOS 2.0 and later.

See Also
- acceptConnectionInBackgroundAndNotify (page 477)

- readToEndOfFileInBackgroundAndNotifyForModes: (page 484)
- enqueueNotification:postingStyle:coalesceMask:forModes: (page 855) (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 483) for details of this method's operation. The method differs from readToEndOfFileInBackgroundAndNotify (page 483) in that *modes* specifies the run-loop mode (or modes) in which NSFileHandleReadToEndOfFileCompletionNotification (page 490) can be posted.

Availability
Available in iOS 2.0 and later.

See Also
- acceptConnectionInBackgroundAndNotifyForModes: (page 478)
- enqueueNotification:postingStyle:coalesceMask:forModes: (page 855) (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 iOS 2.0 and later.

See Also
- offsetInFile (page 481)

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 iOS 2.0 and later.

See Also
- offsetInFile (page 481)

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 iOS 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 iOS 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 489). After the notification has been posted, the thread is terminated.

Availability

Available in iOS 2.0 and later.

See Also

- waitForDataInBackgroundAndNotifyForModes: (page 486)

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 489). After the notification has been posted, the thread is terminated. This method differs from waitForDataInBackgroundAndNotify (page 486) in that *modes* specifies the run-loop mode (or modes) in which NSFileHandleDataAvailableNotification (page 489) can be posted.

Availability

Available in iOS 2.0 and later.

See Also
- `waitForDataInBackgroundAndNotify` (page 486)

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 iOS 2.0 and later.

See Also
- `availableData` (page 478)
- `readDataOfLength:` (page 481)
- `readDataToEndOfFile` (page 482)

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 489) notification.

The corresponding value is the `NSFileHandle` object representing the "near" end of a socket connection.

Available in iOS 2.0 and later.

Declared in `NSFileHandle.h`.

`NSFileHandleNotificationDataItem`
A key in the userinfo dictionary in a NSFileHandleReadCompletionNotification (page 489) and NSFileHandleReadToEndOfFileCompletionNotification (page 490).

The corresponding value is an `NSData` object containing the available data read from a socket connection.

Available in iOS 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 iOS 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 iOS 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 477) or `acceptConnectionInBackgroundAndNotifyForModes:` (page 478) 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 iOS 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 486) or `waitForDataInBackgroundAndNotifyForModes:` (page 486) 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 iOS 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 482) or `readInBackgroundAndNotifyForModes:` (page 483) 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 iOS 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 483) or `readToEndOfFileInBackgroundAndNotifyForModes:` (page 484) 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSFileManager.h
Companion guide	Low-Level File Management Programming Topics

Overview

The `NSFileManager` class enables you to perform many generic file-system operations and insulates an application from the underlying file system.

In iOS and Mac OS X v 10.5 and later you should consider using `[[NSFileManager alloc] init]` rather than the singleton method `defaultManager`. Instances of `NSFileManager` are considered thread-safe when created using `[[NSFileManager alloc] init]`.

Tasks

Creating a File Manager

`- init` (page 519)
Returns an initialized `NSFileManager` instance.

`+ defaultManager` (page 497)
Returns the default `NSFileManager` object for the file system.

Moving an Item

`- fileManager:shouldMoveItemAtPath:toPath:` (page 538) *delegate method*
An `NSFileManager` object sends this message immediately before attempting to move to a given path.

- `fileManager:shouldMoveItemAtURL:toURL:` (page 538) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to move to a given URL.
- `moveItemAtPath:toPath:error:` (page 525)
  Moves the directory or file specified by a given path to a different location in the file system identified by another path.
- `moveItemAtURL:toURL:error:` (page 526)
  Moves the directory or file specified by a given URL to a different location in the file system identified by another URL.
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 543) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given path.
- `fileManager:shouldProceedAfterError:movingItemAtURL:toURL:` (page 544) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to move to a given URL.

Copying an Item

- `fileManager:shouldCopyItemAtPath:toPath:` (page 535) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to copy to a given path.
- `fileManager:shouldCopyItemAtURL:toURL:` (page 536) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to copy to a given URL.
- `copyItemAtPath:toPath:error:` (page 503)
  Copies the directory or file specified in a given path to a different location in the file system identified by another path.
- `copyItemAtURL:toURL:error:` (page 504)
  Copies the directory or file specified in a given URL to a different location in the file system identified by another URL.
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 540) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given path.
- `fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:` (page 541) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given URL.

Removing an Item

- `fileManager:shouldRemoveItemAtPath:` (page 546) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to delete an item at a given path.
- `fileManager:shouldRemoveItemAtURL:` (page 546) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to delete an item at a given URL.

- `removeItemAtPath:error:` (page 527)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given path.
- `removeItemAtURL:error:` (page 528)
  Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given URL.
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to delete a given path.
- `fileManager:shouldProceedAfterError:removingItemAtURL:` (page 545) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to delete a given URL.

Creating an Item

- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506)
  Creates a directory with given attributes at a specified path.
- `createFileAtPath:contents:attributes:` (page 507)
  Creates a file at a given path that has given attributes and contents.
- `createDirectoryAtPath:attributes:` (page 505) Deprecated in iOS 2.0
  Creates a directory (without contents) at a given path with given attributes. (Deprecated. Use `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506) instead.)

Linking an Item

- `fileManager:shouldLinkItemAtPath:toPath:` (page 536) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to link to a given path.
- `fileManager:shouldLinkItemAtURL:toURL:` (page 537) *delegate method*
  An `NSFileManager` object sends this message immediately before attempting to link to a given URL.
- `linkItemAtPath:toPath:error:` (page 522)
  Creates a hard link from a source to a destination identified by a path.
- `linkItemAtURL:toURL:error:` (page 523)
  Creates a hard link from a source to a destination identified by a URL.
- `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 542) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to hard-link to a given path.
- `fileManager:shouldProceedAfterError:linkingItemAtURL:toURL:` (page 542) *delegate method*
  An `NSFileManager` object sends this message if an error occurs during an attempt to hard-link to a given URL.

Symbolic-Link Operations

- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 508)
  Creates a symbolic link identified by a given path that refers to a given location.
- `destinationOfSymbolicLinkAtPath:error:` (page 510)
  Returns the path of the item pointed to by a symbolic link.
- `createSymbolicLinkAtPath:pathContent:` (page 507) Deprecated in iOS 2.0
  Creates a symbolic link identified by a given path that refers to a given location. (Deprecated. Use `createSymbolicLinkAtPath:withDestinationPath:error:` (page 508) instead.)
- `pathContentOfSymbolicLinkAtPath:` (page 527) Deprecated in iOS 2.0
  Returns the path of the directory or file that a symbolic link at a given path refers to. (Deprecated. Use `destinationOfSymbolicLinkAtPath:error:` (page 510) instead.)

Handling File Operations

The methods described in this section are to be implemented by the callback handler passed to several methods of `NSFileManager`. These methods have been deprecated as of Mac OS X 10.5. Use the corresponding delegate methods instead.

- `fileManager:shouldProceedAfterError:` (page 539) *delegate method* Deprecated in iOS 2.0
  An `NSFileManager` object sends this message to its handler for each error it encounters when copying, moving, removing, or linking files or directories. (Deprecated. See delegate methods for copy, move, remove, and link methods.)
- `fileManager:willProcessPath:` (page 547) *delegate method* Deprecated in iOS 2.0
  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. (Deprecated. See delegate methods for copy, move, link, and remove methods.)

Getting and Comparing File Contents

- `contentsAtPath:` (page 500)
  Returns as an `NSData` object the contents of the file at at given path.
- `contentsEqualAtPath:andPath:` (page 501)
  Returns a Boolean value that indicates whether the files or directories in specified paths have the same contents.

Discovering Directory Contents

- `mountedVolumeURLsIncludingResourceValuesForKeys:options:` (page 524)
  Returns the mounted volumes available on the computer.
- `contentsOfDirectoryAtURL:includingPropertiesForKeys:options:error:` (page 502)
  Returns the contents of a directory.
- `contentsOfDirectoryAtPath:error:` (page 501)
  Returns the directories and files (including symbolic links) contained in a given directory.

- `enumeratorAtPath:` (page 512)
  Creates and returns an `NSDirectoryEnumerator` object that enumerates the contents of the directory at a given path.
- `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` (page 513)
  Creates and returns a directory enumerator object that enumerates the contents of the directory at a given URL.
- `subpathsAtPath:` (page 532)
  Returns an array that contains (as `NSString` objects) the contents of the directory identified by a given path.
- `subpathsOfDirectoryAtPath:error:` (page 533)
  Returns an array that contains the filenames of the items in the directory specified by a given path and all its subdirectories recursively.
- `directoryContentsAtPath:` (page 510) Deprecated in iOS 2.0
  Returns the directories and files (including symbolic links) contained in a given directory. (Deprecated. Use `contentsOfDirectoryAtPath:error:` (page 501) instead.)

Determining Access to Files

- `fileExistsAtPath:` (page 516)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `fileExistsAtPath:isDirectory:` (page 517)
  Returns a Boolean value that indicates whether a file or directory exists at a specified path.
- `isReadableFileAtPath:` (page 520)
  Returns a Boolean value that indicates whether the invoking object appears able to read a specified file.
- `isWritableFileAtPath:` (page 521)
  Returns a Boolean value that indicates whether the invoking object appears able to write to a specified file.
- `isExecutableFileAtPath:` (page 520)
  Returns a Boolean value that indicates whether the operating system appears able to execute a specified file.
- `isDeletableFileAtPath:` (page 519)
  Returns a Boolean value that indicates whether the invoking object appears able to delete a specified file.

Getting and Setting Attributes

- `componentsToDisplayForPath:` (page 500)
  Returns an array of `NSString` objects representing the user-visible components of a given path.
- `displayNameAtPath:` (page 511)
  Returns the name of the file or directory at a given path in a localized form appropriate for presentation to the user.
- `attributesOfItemAtPath:error:` (page 498)
  Returns the attributes of the item at a given path.

- `attributesOfFileSystemForPath:error:` (page 497)
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides.
- `setAttributes:ofItemAtPath:error:` (page 530)
  Sets the attributes of a given file or directory.
- `changeFileAttributes:atPath:` (page 499) Deprecated in iOS 2.0
  Changes the attributes of a given file or directory. (Deprecated. Use `setAttributes:ofItemAtPath:error:` (page 530) instead.)
- `fileAttributesAtPath:traverseLink:` (page 515) Deprecated in iOS 2.0
  Returns a dictionary that describes the POSIX attributes of the file specified at a given. (Deprecated. Use `attributesOfItemAtPath:error:` (page 498) instead.)
- `fileSystemAttributesAtPath:` (page 518) Deprecated in iOS 2.0
  Returns a dictionary that describes the attributes of the mounted file system on which a given path resides. (Deprecated. Use `attributesOfFileSystemForPath:error:` (page 497) instead.)

Getting Representations of File Paths

- `fileSystemRepresentationWithPath:` (page 518)
  Returns a C-string representation of a given path that properly encodes Unicode strings for use by the file system.
- `stringWithFileSystemRepresentation:length:` (page 532)
  Returns an `NSString` object converted from the C-string representation of a pathname in the current file system.

Managing the Delegate

- `setDelegate:` (page 531)
  Sets the delegate for the receiver.
- `delegate` (page 509)
  Returns the delegate for the receiver.

Managing the Current Directory

- `changeCurrentDirectoryPath:` (page 499)
  Changes the path of the current directory for the current process to a given path.
- `currentDirectoryPath` (page 509)
  Returns the path of the program's current directory.

Locating System Directories

- `URLForDirectory:inDomain:appropriateForURL:create:error:` (page 534)
  Locates and optionally creates the specified common directory in a domain.
- `URLsForDirectory:inDomains:` (page 534)
  Returns an array of URLs for the specified common directory in the requested domains.

Safely Replace a File

`- replaceItemAtURL:withItemAtURL:backupItemName:options:resultingItemURL:error:` (page 529)

Replaces the contents specified by the first URL with the contents of the second URL in a manner that insures no data loss occurs.

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
This will always return the same instance of the file manager. The returned object is not thread safe.

In Mac OS X v 10.5 and later you should consider using `[[NSFileManager alloc] init]` rather than the singleton method `defaultManager`. Using `[[NSFileManager alloc] init]` instead, the resulting `NSFileManager` instance is thread safe.

Availability
Available in iOS 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 553) for a description of the keys available in the dictionary.

Discussion
This method does not traverse a terminal symbolic link.

Availability
Available in iOS 2.0 and later.

See Also
- `attributesOfItemAtPath:error:` (page 498)
- `setAttributes:ofItemAtPath:error:` (page 530)

Declared In
`NSFileManager.h`

attributesOfItemAtPath:error:

Returns 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 549).

Special Considerations

As a convenience, `NSDictionary` provides a set of methods (declared as a category on `NSDictionary`) for quickly and efficiently obtaining attribute information from the returned dictionary: `fileGroupOwnerAccountName` (page 400), `fileModificationDate` (page 402), `fileOwnerAccountName` (page 402), `filePosixPermissions` (page 403), `fileSize` (page 403), `fileSystemFileNumber` (page 404), `fileSystemNumber` (page 404), and `fileType` (page 405).

In Mac OS X v 10.6 and earlier, if the last component of the path is a symbolic link (the value of the `NSFileType` key in the attributes dictionary is `NSFileTypeSymbolicLink`), it will not be traversed. This behavior may change in a future version of the Mac OS X.

Availability
Available in iOS 2.0 and later.

See Also
- `setAttributes:ofItemAtPath:error:` (page 530)

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 iOS 2.0 and later.

See Also

- currentDirectoryPath (page 509)
- fileExistsAtPath:isDirectory: (page 517)
- contentsOfDirectoryAtPath:error: (page 501)

Declared In

NSFileManager.h

changeFileAttributes:atPath:

Changes the attributes of a given file or directory. (Deprecated in iOS 2.0. Use setAttributes:ofItemAtPath:error: (page 530) instead.)

```
- (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

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `setAttributes:ofItemAtPath:error:` (page 530) instead.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 2.0.

See Also
- `attributesOfItemAtPath:error:` (page 498)
- `setAttributes:ofItemAtPath:error:` (page 530)

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`*. Returns `nil` if path does not exist.

Discussion
These components cannot be used for path operations and are only suitable for display to the user.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also

- `contentsEqualAtPath:andPath:` (page 501)
- `createFileAtPath:contents:attributes:` (page 507)

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 iOS 2.0 and later.

See Also

- `contentsAtPath:` (page 500)

Declared In

`NSFileManager.h`

contentsOfDirectoryAtPath:error:

Returns the directories and files (including symbolic links) contained in a given directory.

```
- (NSArray *)contentsOfDirectoryAtPath:(NSString *)path error:(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 iOS 2.0 and later.

See Also

- `currentDirectoryPath` (page 509)
- `fileExistsAtPath:isDirectory:` (page 517)
- `enumeratorAtPath:` (page 512)
- `subpathsAtPath:` (page 532)

Declared In

`NSFileManager.h`

contentsOfDirectoryAtURL:includingPropertiesForKeys:options:error:

Returns the contents of a directory.

```
- (NSArray *)contentsOfDirectoryAtURL:(NSURL *)url
    includingPropertiesForKeys:(NSArray *)keys
    options:(NSDirectoryEnumerationOptions)mask error:(NSError **)error
```

Parameters

url

The location of the directory for which you want an enumeration.

keys

On input, an array of property keys for which you would like the corresponding values. These specify the file properties that are pre-fetched for each of the files in the directory.

When an array is specified for this parameter, the specified property values are pre-fetched and cached with each enumerated URL. The property keys that can be requested are listed in `Common File System Resource Keys` (page 1399)).

mask

Options for the enumeration. Because this method only performs shallow enumeration only the `NSDirectoryEnumerationSkipsHiddenFiles` (page 548) option should be used.

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 `NSURL` objects identifying the directory entries. If the directory contains no entries, this method returns an empty array. If an error occurs, returns `nil` after setting `error` to an `NSError` object that describes the reason why the directory could not be enumerated.

Discussion
This method always does a shallow enumeration of the specified directory (i.e. it always acts as if `NSDirectoryEnumerationSkipsSubdirectoryDescendants` (page 548) has been specified). If you need to perform a deep enumeration, use `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` (page 513)].

The order of the files within the returned array is undefined.

Availability
Available in iOS 4.0 and later.

See Also
`- contentsOfDirectoryAtPath:error:` (page 501)

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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 540) message, `copyItemAtPath:toPath:error:` also returns `YES`. Otherwise this method returns `NO`. The method also attempts to make the attributes of the directory or file at *dstPath* identical to *srcPath*, but ignores any failure at this attempt.

Discussion
If *srcPath* is a file, the method creates a file at *dstPath* that holds the exact contents of the original file (this includes BSD special files). If *srcPath* is a directory, the method creates a new directory at *dstPath* and recursively populates it with duplicates of the files and directories contained in *srcPath*, preserving all

links. The file specified in *srcPath* must exist, while *dstPath* must not exist prior to the operation. When a file is being copied, the destination path must end in a filename—there is no implicit adoption of the source filename. Symbolic links are not traversed but are themselves copied. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also copied.

`NSFileManager` sends your delegate `fileManager:shouldCopyItemAtPath:toPath:` (page 535) when it begins a copy operation. If the delegate returns `YES`, `NSFileManager` attempts to copy the item. If the delegate returns `NO`, the `copyItemAtPath:toPath:error:` function does not copy the item.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 540) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `copyItemAtPath:toPath:error:` function terminates and passes the error back in the `error` parameter.

Availability
Available in iOS 2.0 and later.

See Also
- `copyItemAtURL:toURL:error:` (page 504)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 535)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 540)
- `linkItemAtPath:toPath:error:` (page 522)
- `moveItemAtPath:toPath:error:` (page 525)
- `removeItemAtPath:error:` (page 527)

Declared In
`NSFileManager.h`

copyItemAtURL:toURL:error:

Copies the directory or file specified in a given URL to a different location in the file system identified by another URL.

```
- (BOOL)copyItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL error:(NSError **)error
```

Parameters

srcURL
The URL of a file or directory.

dstURL
The URL 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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:` (page 541) message, `copyItemAtURL:toURL:error:` also returns `YES`. Otherwise this method returns `NO`. The method also attempts to make the attributes of the directory or file at *dstURL* identical to *srcURL*, but ignores any failure at this attempt.

Discussion
If *srcURL* is a file, the method creates a file at *dstURL* that holds the exact contents of the original file (this includes BSD special files). If *srcURL* is a directory, the method creates a new directory at *dstURL* and recursively populates it with duplicates of the files and directories contained in *srcURL*, preserving all links. The file specified in *srcURL* must exist, while *dstURL* must not exist prior to the operation. When a file is being copied, the destination URL must end in a filename—there is no implicit adoption of the source filename. Symbolic links are not traversed but are themselves copied. File or directory attributes—that is, metadata such as owner and group numbers, file permissions, and modification date—are also copied.

`NSFileManager` sends your delegate `fileManager:shouldCopyItemAtURL:toURL:` (page 536) when it begins a copy operation. If the delegate returns `YES`, `NSFileManager` attempts to copy the item. If the delegate returns `NO`, the `copyItemAtURL:toURL:error:` function does not copy the item.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:` (page 541) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `copyItemAtURL:toURL:error:` function terminates and passes the error back in the `error` parameter.

Availability
Available in iOS 4.0 and later.

See Also
- `copyItemAtPath:toPath:error:` (page 503)
- `fileManager:shouldCopyItemAtURL:toURL:` (page 536)
- `fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:` (page 541)
- `linkItemAtPath:toPath:error:` (page 522)
- `moveItemAtPath:toPath:error:` (page 525)
- `removeItemAtPath:error:` (page 527)

Declared In
`NSFileManager.h`

createDirectoryAtPath:attributes:

Creates a directory (without contents) at a given path with given attributes. (Deprecated in iOS 2.0. Use `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506) instead.)

```
- (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 547) 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 or the directory already exists, otherwise `NO`.

Special Considerations

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506) instead.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 2.0.

See Also
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506)
- `changeCurrentDirectoryPath:` (page 499)
- `setAttributes:ofItemAtPath:error:` (page 530)
- `createFileAtPath:contents:attributes:` (page 507)
- `currentDirectoryPath` (page 509)

Declared In
`NSFileManager.h`

createDirectoryAtPath:withIntermediateDirectories:attributes:error:

Creates a directory with given attributes at a specified path.

```
- (BOOL)createDirectoryAtPath:(NSString *)path
    withIntermediateDirectories:(BOOL)createIntermediates attributes:(NSDictionary
     *)attributes error:(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 fails if any parent of the directory to be created does not exist. In addition, if you pass `NO` for this parameter, the directory must not exist at the time this call is made.

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 547) 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 or already exists, otherwise `NO`.

Availability
Available in iOS 2.0 and later.

See Also
- `changeCurrentDirectoryPath:` (page 499)
- `setAttributes:ofItemAtPath:error:` (page 530)
- `createFileAtPath:contents:attributes:` (page 507)
- `currentDirectoryPath` (page 509)

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 549) 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`.

Discussion
If a file already exists at *path*, then if the file can be overwritten (subject to user privileges) it will be.

Availability
Available in iOS 2.0 and later.

See Also
- `contentsAtPath:` (page 500)
- `setAttributes:ofItemAtPath:error:` (page 530)
- `setAttributes:ofItemAtPath:error:` (page 530)
- `attributesOfItemAtPath:error:` (page 498)

Declared In
`NSFileManager.h`

createSymbolicLinkAtPath:pathContent:

Creates a symbolic link identified by a given path that refers to a given location. (Deprecated in iOS 2.0. Use `createSymbolicLinkAtPath:withDestinationPath:error:` (page 508) instead.)

```
- (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

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use createSymbolicLinkAtPath:withDestinationPath:error: (page 508) instead.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also

- createSymbolicLinkAtPath:withDestinationPath:error: (page 508)
- destinationOfSymbolicLinkAtPath:error: (page 510)
- linkItemAtPath:toPath:error: (page 522)

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 *)path withDestinationPath:(NSString
    *)destPath error:(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 a terminal symbolic link.

Availability
Available in iOS 2.0 and later.

See Also
- `destinationOfSymbolicLinkAtPath:error:` (page 510)
- `linkItemAtPath:toPath:error:` (page 522)

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 499).

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 iOS 2.0 and later.

See Also
- `changeCurrentDirectoryPath:` (page 499)
- `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506)

Declared In
`NSFileManager.h`

delegate

Returns the delegate for the receiver.

```
- (id)delegate
```

Return Value
The delegate for the receiver.

Availability
Available in iOS 2.0 and later.

Declared In
`NSFileManager.h`

destinationOfSymbolicLinkAtPath:error:

Returns the path of the item pointed to by a symbolic link.

```
- (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.

Availability

Available in iOS 2.0 and later.

See Also

`- createSymbolicLinkAtPath:withDestinationPath:error:` (page 508)

Declared In

`NSFileManager.h`

directoryContentsAtPath:

Returns the directories and files (including symbolic links) contained in a given directory. (Deprecated in iOS 2.0. Use `contentsOfDirectoryAtPath:error:` (page 501) instead.)

```
- (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

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `contentsOfDirectoryAtPath:error:` (page 501) instead.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also
- `contentsOfDirectoryAtPath:error:` (page 501)
- `currentDirectoryPath` (page 509)
- `fileExistsAtPath:isDirectory:` (page 517)
- `enumeratorAtPath:` (page 512)
- `subpathsAtPath:` (page 532)

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` as is.

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 alloc] init];
    NSString *displayNameAtPath = [fileManager
displayNameAtPath:documentsDirectory];
    NSLog(@"%@", displayNameAtPath);
    [fileManager release];
}
```

Availability

Available in iOS 2.0 and later.

See Also

`lastPathComponent` (page 1245) (`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 filename, the method returns an enumerator object that enumerates no files—the first call to `nextObject` (page 424) 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. 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 *docsDir = [NSHomeDirectory() stringByAppendingPathComponent:
@"Documents"];
NSFileManager *localFileManager=[[NSFileManager alloc] init];
NSDirectoryEnumerator *dirEnum =
    [localFileManager enumeratorAtPath:docsDir];

NSString *file;
while (file = [dirEnum nextObject]) {
    if ([[file pathExtension] isEqualToString: @"doc"]) {
        // process the document
        [self scanDocument: [docsDir stringByAppendingPathComponent:file]];
    }
}
[localFileManager release];
```

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 iOS 2.0 and later.

See Also

- `currentDirectoryPath` (page 509)
- `attributesOfItemAtPath:error:` (page 498)
- `contentsOfDirectoryAtPath:error:` (page 501)
- `subpathsAtPath:` (page 532)
- `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` (page 513)

Declared In

`NSFileManager.h`

enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:

Creates and returns a directory enumerator object that enumerates the contents of the directory at a given URL.

```
- (NSDirectoryEnumerator *)enumeratorAtURL:(NSURL *)url
    includingPropertiesForKeys:(NSArray *)keys
    options:(NSDirectoryEnumerationOptions)mask errorHandler:(BOOL (^)(NSURL *url,
     NSError *error))handler
```

Parameters

url

The location of the directory for which you want an enumeration.

keys

On input, an array of property keys for which you would like the corresponding values. Specify `NULL` for this array if you do not want any property values. The property keys that can be requested are listed in `Common File System Resource Keys` (page 1399).

When an array is specified for this parameter, the specified property values are pre-fetched and cached with each enumerated URL.

mask

Options for the enumeration. See "Directory Enumeration Options" (page 548).

handler

The optional `'errorHandler'` block argument is invoked when an error occurs. Parameters to the block are the URL on which an error occurred and the error. When the error handler returns `YES`, enumeration continues if possible. Enumeration stops immediately when the error handler returns `NO`.

Return Value

An `NSDirectoryEnumerator` object that enumerates the contents of the directory at *url*. If *url* 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 *url* is a filename, the method returns an enumerator object that enumerates no files—the first call to `nextObject` (page 424) returns `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.

The `NSDirectoryEnumerator` class has methods for skipping descendants of the existing path and for returning the number of levels deep the current object is in the directory hierarchy being enumerated (where the directory passed to `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` is considered to be level 0).

This code fragment enumerates the a URL and it's subdirectories, collecting the URLs of files (skips directories). It also demonstrates how to ignore the contents of specified directories, in this case directories named "_extras"

```
-(void)scanURLIgnoringExtras:(NSURL *)directoryToScan
{
    // Create a local file manager instance
```

```
    NSFileManager *localFileManager=[[NSFileManager alloc] init];

    // Enumerate the directory (specified elsewhere in your code)
    // Request the two properties the method uses, name and isDirectory
    // Ignore hidden files
    // The errorHandler: parameter is set to nil. Typically you'd want to present a
panel
    NSDirectoryEnumerator *dirEnumerator = [localFileManager
enumeratorAtURL:directoryToScan
                                                    includingPropertiesForKeys:[NSArray
arrayWithObjects:NSURLNameKey,

NSURLIsDirectoryKey,nil]

options:NSDirectoryEnumerationSkipsHiddenFiles
                                                                        errorHandler:nil];

    // An array to store the all the enumerated file names in
    NSMutableArray *theArray=[NSMutableArray array];

    // Enumerate the dirEnumerator results, each value is stored in allURLs
    for (NSURL *theURL in dirEnumerator) {

        // Retrieve the file name. From NSURLNameKey, cached during the enumeration.
        NSString *fileName;
        [theURL getResourceValue:&fileName forKey:NSURLNameKey error:NULL];

        // Retrieve whether a directory. From NSURLIsDirectoryKey, also cached during
the enumeration.
        NSNumber *isDirectory;
        [theURL getResourceValue:&isDirectory forKey:NSURLIsDirectoryKey error:NULL];

        // Ignore files under the _extras directory
       if (([fileName caseInsensitiveCompare:@"_extras"]==NSOrderedSame) && ([isDirectory
 boolValue]==YES))
        {
            [dirEnumerator skipDescendants];
        }
        else
        {
            // Add full path for non directories
            if ([isDirectory boolValue]==NO)
                [theArray addObject:theURL];

        }
    }

    // Do something with the path URLs.
    NSLog(@"theArray - %@",theArray);

    // Release the localFileManager.
    [localFileManager release];

}
```

Availability
Available in iOS 4.0 and later.

See Also
- `enumeratorAtPath:` (page 512)

Declared In
`NSFileManager.h`

fileAttributesAtPath:traverseLink:

Returns a dictionary that describes the POSIX attributes of the file specified at a given. (Deprecated in iOS 2.0. Use `attributesOfItemAtPath:error:` (page 498) instead.)

```
- (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 549). 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 alloc] init];
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);
}
[fileManager release];
```

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 400), `fileModificationDate` (page 402), `fileOwnerAccountName` (page 402), `filePosixPermissions` (page 403), `fileSize` (page 403), `fileSystemFileNumber` (page 404), `fileSystemNumber` (page 404), and `fileType` (page 405). 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

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `attributesOfItemAtPath:error:` (page 498) instead.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also

- `attributesOfItemAtPath:error:` (page 498)
- `setAttributes:ofItemAtPath:error:` (page 530)

Declared In

`NSFileManager.h`

fileExistsAtPath:

Returns a Boolean value that indicates whether a file or directory exists at a specified 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 1267), or this method returns `NO`.

Return Value

`YES` if a file specified in *path* exists, otherwise `NO`. If the final element in *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.

Discussion

Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability

Available in iOS 2.0 and later.

See Also

- `fileExistsAtPath:isDirectory:` (page 517)

Declared In
NSFileManager.h

fileExistsAtPath:isDirectory:

Returns a Boolean value that indicates whether a file or directory exists at a specified path.

```
- (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 1267), 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 the final element in *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 alloc] init];
    NSString *fontPath = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Fonts"];

    if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir) {
        subpaths = [fileManager subpathsAtPath:fontPath];
// ...
[fileManager release];
```

Note: Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability
Available in iOS 2.0 and later.

See Also
- `fileExistsAtPath:` (page 516)

Declared In
`NSFileManager.h`

fileSystemAttributesAtPath:

Returns a dictionary that describes the attributes of the mounted file system on which a given path resides. (Deprecated in iOS 2.0. Use `attributesOfFileSystemForPath:error:` (page 497) instead.)

```
- (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 553) for a description of the keys available in the dictionary.

Special Considerations

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `attributesOfFileSystemForPath:error:` (page 497) instead.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 2.0.

See Also
- `attributesOfFileSystemForPath:error:` (page 497)
- `attributesOfItemAtPath:error:` (page 498)
- `setAttributes:ofItemAtPath:error:` (page 530)

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 iOS 2.0 and later.

See Also

`- stringWithFileSystemRepresentation:length:` (page 532)

Declared In

`NSFileManager.h`

init

Returns an initialized `NSFileManager` instance.

```
- init
```

Return Value

An initialized `NSFileManager` instance.

Discussion

In Mac OS X v 10.4 and earlier sending the `init` message was undefined. In iOS and Mac OS X 10.5 and later it will initialize the receiver.

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.

Note: Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability
Available in iOS 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.

Note: Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability
Available in iOS 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.

Note: Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability

Available in iOS 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.

Note: Attempting to predicate behavior based on the current state of the file system or a particular file on the file system is not recommended. Doing so can cause odd behavior in the case of file system race conditions. It's far better to attempt an operation (such as loading a file or creating a directory), check for errors, and handle any error gracefully than it is to try to figure out ahead of time whether the operation will succeed. For more information on file system race conditions, see "Avoiding Race Conditions and Insecure File Operations" in *Secure Coding Guide*.

Availability
Available in iOS 2.0 and later.

Declared In
`NSFileManager.h`

linkItemAtPath:toPath:error:

Creates a hard link from a source to a destination identified by a path.

```
- (BOOL)linkItemAtPath:(NSString *)srcPath toPath:(NSString *)dstPath error:(NSError **)error
```

Parameters

srcPath

A path that identifies a source file.

The file or link specified by *srcPath* must exist. *srcPath* must not identify a directory.

dstPath

A path that identifies a destination file or directory on the same filesystem as *srcPath*.

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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:` (page 542) message, `linkItemAtPath:toPath:error:` also returns `YES`. Otherwise this method returns `NO`.

Discussion
If pathname *srcPath* identifies a file, this method hard-links the file specified in *dstPath* to it.

Amongst other reasons (such as the disk being full, permissions problems, and so on), this method will fail if:

- *srcPath* doesn't point to any file in the file system;
- *srcPath* points to an existing symbolic link, but the symbolic link is "broken" (it doesn't in turn point to an existing regular file in the file system);
- *srcPath* points to a directory;
- The computer has more than one file system (such as extra partitions, mounted disk images, or network volumes), and *srcPath* and *dstPath* specify paths in different file systems.

NSFileManager sends your delegate fileManager:shouldLinkItemAtPath:toPath: (page 536) when it begins a hard-link operation. If the delegate returns YES, NSFileManager attempts to hard-link the item. If the delegate returns NO, the linkItemAtPath:toPath:error: function does not hard-link the item.

NSFileManager sends your delegate fileManager:shouldProceedAfterError:linkingItemAtPath:toPath: (page 542) when it encounters any error in processing. If the delegate returns YES, then NSFileManager proceeds as if no error had occurred. If it returns NO, the linkItemAtPath:toPath:error: function terminates and passes the error back in the error parameter.

Availability
Available in iOS 2.0 and later.

See Also
- linkItemAtURL:toURL:error: (page 523)
- fileManager:shouldLinkItemAtPath:toPath: (page 536)
- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath: (page 542)
- createSymbolicLinkAtPath:withDestinationPath:error: (page 508)
- copyItemAtPath:toPath:error: (page 503)
- moveItemAtPath:toPath:error: (page 525)
- removeItemAtPath:error: (page 527)

Declared In
NSFileManager.h

linkItemAtURL:toURL:error:

Creates a hard link from a source to a destination identified by a URL.

```
- (BOOL)linkItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL error:(NSError **)error
```

Parameters

srcURL

A URL that identifies a source file.

The file or link specified by *srcURL* must exist. *srcURL* must not identify a directory.

dstURL

A URL that identifies a destination file or directory on the same filesystem as *srcURL*.

The destination should not yet exist. The destination URL 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. If the operation is not successful, but the delegate returns YES from the fileManager:shouldProceedAfterError:linkingItemAtURL:toURL: (page 542) message, linkItemAtURL:toURL:error: also returns YES. Otherwise this method returns NO.

Discussion
If *srcURL* identifies a file, this method hard-links the file specified in *dstURL* to it. If *srcURL* is a symbolic link, this method copies it to *dstURL* instead of creating a hard link. Symbolic links in *srcURL* are not traversed.

Amongst other reasons (such as the disk being full, permissions problems, and so on), this method will fail if:

- *srcURL* doesn't point to any file in the file system;
- *srcURL* points to an existing symbolic link, but the symbolic link is “broken" (it doesn't in turn point to an existing regular file in the file system);
- *srcURL* points to a directory;
- The computer has more than one file system (such as extra partitions, mounted disk images, or network volumes), and *srcURL* and *dstURL* specify URLs in different file systems.

`NSFileManager` sends your delegate `fileManager:shouldLinkItemAtURL:toURL:` (page 537) when it begins a hard-link operation. If the delegate returns `YES`, `NSFileManager` attempts to hard-link the item. If the delegate returns `NO`, the `linkItemAtURL:toURL:error:` function does not hard-link the item.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:linkingItemAtURL:toURL:` (page 542) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `linkItemAtURL:toURL:error:` function terminates and passes the error back in the `error` parameter.

Availability
Available in iOS 4.0 and later.

See Also
- `linkItemAtPath:toPath:error:` (page 522)
- `fileManager:shouldLinkItemAtURL:toURL:` (page 537)
- `fileManager:shouldProceedAfterError:linkingItemAtURL:toURL:` (page 542)
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 508)
- `copyItemAtURL:toURL:error:` (page 504)
- `moveItemAtURL:toURL:error:` (page 526)
- `removeItemAtURL:error:` (page 528)

Declared In
`NSFileManager.h`

mountedVolumeURLsIncludingResourceValuesForKeys:options:

Returns the mounted volumes available on the computer.

```
- (NSArray *)mountedVolumeURLsIncludingResourceValuesForKeys:(NSArray *)propertyKeys
    options:(NSVolumeEnumerationOptions)options
```

Parameters

propertyKeys

On input, an array of property keys for which the corresponding resource values should be pre-fetched. Specify `NULL` for this array if you do not want any resource values. The property keys that can be requested are listed in `Common File System Resource Keys` (page 1399).

options

Option flags for the enumeration. See “Mounted Volume Enumeration Options” (page 547).

Return Value
An array of `NSURL` objects identifying the mounted volumes.

Discussion
This call may block if I/O is required to determine values for the requested *`propertyKeys`*.

Availability
Available in iOS 4.0 and later.

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 *)srcPath toPath:(NSString *)dstPath error:(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 to be moved. *`destination`* must not yet exist. The destination path must end in a directory name or filename; there is no implicit adoption of the source name.

`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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 543) message, `moveItemAtPath:toPath:error:` also returns `YES`. Otherwise this method returns `NO`.

Discussion
If the source path and the destination path are not on the same device, `NSFileManager` performs a copy to the destination path and removes the original. If the copy does not succeed, this method returns an error and `NSFileManager` removes the incomplete copy, leaving the original in place.

`NSFileManager` sends your delegate `fileManager:shouldMoveItemAtPath:toPath:` (page 538) when it begins a move operation. If the delegate returns `YES`, `NSFileManager` attempts to move the item. If the delegate returns `NO`, the `moveItemAtPath:toPath:error:` function does not move the item.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 543) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `moveItemAtPath:toPath:error:` function terminates and passes the error back in the `error` parameter.

Availability
Available in iOS 2.0 and later.

See Also
- `moveItemAtURL:toURL:error:` (page 526)
- `fileManager:shouldMoveItemAtPath:toPath:` (page 538)
- `copyItemAtPath:toPath:error:` (page 503)
- `linkItemAtPath:toPath:error:` (page 522)
- `removeItemAtPath:error:` (page 527)

Declared In
`NSFileManager.h`

moveItemAtURL:toURL:error:

Moves the directory or file specified by a given URL to a different location in the file system identified by another URL.

```
- (BOOL)moveItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL error:(NSError **)error
```

Parameters

srcURL
The URL of a file or directory to move. *srcURL* must exist.

dstURL
The URL to which the file or directory at *srcURL* is to be moved. *destination* must not yet exist. The destination URL must end in a directory name or filename; there is no implicit adoption of the source name.

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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:movingItemAtURL:toURL:` (page 544) message, `moveItemAtURL:toURL:error:` also returns `YES`. Otherwise this method returns `NO`.

Discussion
If the source path and the destination path are not on the same device, `NSFileManager` performs a copy to the destination path and removes the original. If the copy does not succeed, this method returns an error and `NSFileManager` removes the incomplete copy, leaving the original in place.

`NSFileManager` sends your delegate `fileManager:shouldMoveItemAtURL:toURL:` (page 538) when it begins a move operation. If the delegate returns `YES`, `NSFileManager` attempts to move the item. If the delegate returns `NO`, the `moveItemAtURL:toURL:error:` function does not move the item.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:movingItemAtURL:toURL:` (page 544) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `moveItemAtURL:toURL:error:` function terminates and passes the error back in the `error` parameter.

Availability
Available in iOS 4.0 and later.

See Also
- `moveItemAtPath:toPath:error:` (page 525)
- `fileManager:shouldMoveItemAtURL:toURL:` (page 538)
- `copyItemAtURL:toURL:error:` (page 504)
- `linkItemAtURL:toURL:error:` (page 523)
- `removeItemAtURL:error:` (page 528)

Declared In
`NSFileManager.h`

pathContentOfSymbolicLinkAtPath:

Returns the path of the directory or file that a symbolic link at a given path refers to. (Deprecated in iOS 2.0. Use `destinationOfSymbolicLinkAtPath:error:` (page 510) instead.)

```
- (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

Because this method does not return error information, it has been deprecated as of Mac OS X v10.5. Use `destinationOfSymbolicLinkAtPath:error:` (page 510) instead.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also
- `destinationOfSymbolicLinkAtPath:error:` (page 510)
- `createSymbolicLinkAtPath:withDestinationPath:error:` (page 508)

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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544) message, `removeItemAtPath:error:` also returns `YES`. Otherwise this method returns `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.

`NSFileManager` sends your delegate `fileManager:shouldRemoveItemAtPath:` (page 546) when it begins a delete operation. If the delegate returns `YES`, `NSFileManager` attempts to delete the item. If the delegate returns `NO`, the `removeItemAtPath:error:` function does not delete the item and, if the item is a directory, no children of that item are deleted either.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `removeItemAtPath:error:` function terminates and passes the error back in the `error` parameter.

Availability

Available in iOS 2.0 and later.

See Also

- `removeItemAtURL:error:` (page 528)
- `copyItemAtPath:toPath:error:` (page 503)
- `linkItemAtPath:toPath:error:` (page 522)
- `moveItemAtPath:toPath:error:` (page 525)
- `fileManager:shouldRemoveItemAtPath:` (page 546)
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544)
- `removeItemAtPath:error:` (page 527)

Declared In

`NSFileManager.h`

removeItemAtURL:error:

Deletes the file, link, or directory (including, recursively, all subdirectories, files, and links in the directory) identified by a given URL.

```
- (BOOL)removeItemAtURL:(NSURL *)URL error:(NSError **)error
```

Parameters

URL

The URL 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. If the operation is not successful, but the delegate returns `YES` from the `fileManager:shouldProceedAfterError:removingItemAtURL:` (page 545) message, `removeItemAtURL:error:` also returns `YES`. Otherwise this method returns `NO`.

Discussion

Since the removal of directory contents is so thorough and final, be careful when using this method. If you specify "." or ".." for *URL* an `NSInvalidArgumentException` exception is raised. This method does not traverse symbolic links.

`NSFileManager` sends your delegate `fileManager:shouldRemoveItemAtURL:` (page 546) when it begins a delete operation. If the delegate returns `YES`, `NSFileManager` attempts to delete the item. If the delegate returns `NO`, the `removeItemAtURL:error:` function does not delete the item and, if the item is a directory, no children of that item are deleted either.

`NSFileManager` sends your delegate `fileManager:shouldProceedAfterError:removingItemAtURL:` (page 545) when it encounters any error in processing. If the delegate returns `YES`, then `NSFileManager` proceeds as if no error had occurred. If it returns `NO`, the `removeItemAtURL:error:` function terminates and passes the error back in the `error` parameter.

Availability

Available in iOS 4.0 and later.

See Also

- `removeItemAtPath:error:` (page 527)
- `copyItemAtPath:toPath:error:` (page 503)
- `linkItemAtPath:toPath:error:` (page 522)
- `moveItemAtPath:toPath:error:` (page 525)
- `fileManager:shouldRemoveItemAtPath:` (page 546)
- `fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544)
- `removeItemAtPath:error:` (page 527)

Declared In

`NSFileManager.h`

replaceItemAtURL:withItemAtURL:backupItemName:options:resultingItemURL:error:

Replaces the contents specified by the first URL with the contents of the second URL in a manner that insures no data loss occurs.

```
- (BOOL)replaceItemAtURL:(NSURL *)originalItemURL withItemAtURL:(NSURL *)newItemURL
    backupItemName:(NSString *)backupItemName
   options:(NSFileManagerItemReplacementOptions)options resultingItemURL:(NSURL
   **)resultingURL error:(NSError **)error
```

Parameters

originalItemURL

The item being replaced.

newItemURL

The item which will replace the *originalItemURL*. This item should be placed in a temporary directory as provided by the OS, or in a uniquely named directory placed in the same directory as the original item if the temporary directory is not available.

backupItemName

Optional. If provided, this name will be used to create a backup of the original item.

The backup is placed in the same directory as the original item. If an error occurs during the creation of the backup item, the operation will fail. If there is already an item with the same name as the backup item, that item will be removed.

The backup item will be removed in the event of success unless the `NSFileManagerItemReplacementWithoutDeletingBackupItem` (page 549) option is provided in *options*.

options

Specifies the options used in the replacement. Passing 0 provides the default behavior which uses only the metadata from the new item. The values in "NSFileManagerItemReplacementOptions" (page 548) are also valid and can be combined using the C-bitwise OR operator. Typically 0 is passed.

resultingURL

This URL will be set to the URL which points at the new item. *resultingURL* may be the same as *originalItemURL* if the replacement could be made without having to create a new filesystem object. *resultingURL* may be different than *originalItemURL* if the replacement could not be made without having to create a new object (e.g. going from an rtf document to an rtfd requires the creation of a new item - in this case, *resultingURL* would locate the newly-created rtfd).

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
Returns YES if the replacement was successful, otherwise NO.

Discussion
By default, the creation date, permissions, Finder label and color, and Spotlight comments of the original item will be preserved on the resulting item.

If an error occurs in replacing a filesystem item and the original item has been left in neither the original location nor the temporary location, the NSError returned will contain a user info dictionary with the `NSFileOriginalItemLocationKey` key and its value will be an NSURL instance which locates the item. The error code is one of the various NSFile* errors already present in `NSError Codes` (page 1759).

Availability
Available in iOS 4.0 and later.

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 the following attributes: `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. If the last component of the path is a symbolic link it is traversed.

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 iOS 2.0 and later.

See Also

`- attributesOfItemAtPath:error:` (page 498)

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 iOS 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 iOS 2.0 and later.

See Also

`- fileSystemRepresentationWithPath:` (page 518)

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 alloc] init];
if ([fileManager fileExistsAtPath:fontPath isDirectory:&isDir] && isDir)
    subpaths = [fileManager subpathsAtPath:fontPath];
[fileManager release];
```

Special Considerations

On Mac OS X v10.5 and later, use `subpathsOfDirectoryAtPath:error:` (page 533) instead.

Availability

Available in iOS 2.0 and later.

See Also

- `subpathsOfDirectoryAtPath:error:` (page 533)
- `contentsOfDirectoryAtPath:error:` (page 501)
- `enumeratorAtPath:` (page 512)

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 iOS 2.0 and later.

See Also

- `subpathsAtPath:` (page 532)
- `contentsOfDirectoryAtPath:error:` (page 501)
- `enumeratorAtPath:` (page 512)

Declared In
NSFileManager.h

URLForDirectory:inDomain:appropriateForURL:create:error:

Locates and optionally creates the specified common directory in a domain.

```
- (NSURL *)URLForDirectory:(NSSearchPathDirectory)directory
    inDomain:(NSSearchPathDomainMask)domain appropriateForURL:(NSURL *)url
    create:(BOOL)shouldCreate error:(NSError **)error
```

Parameters

directory

The search path directory. The supported values are described in NSSearchPathDirectory (page 1747).

domain

The domain specifies where the search should occur. The constants are specified by NSSearchPathDomainMask (page 1750). Note: You may not pass the NSAllDomainsMask.

url

If not NULL, and *directory* is NSItemReplacementDirectory (page 1750) the appropriate temporary directory will be located. If the URL is located on another machine, the temporary directory will be on the other machine. If the URL is local, the temporary directory will be local.

shouldCreate

YES if the directory should be created if it doesn't exist, otherwise NO.

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
Returns an NSURL specifying the directory, or nil if an error was encountered.

Discussion
Passing a directory and domain pair that makes no sense (for example NSDesktopDirectory (page 1748) and NSNetworkDomainMask (page 1750)) will raise an exception.

Availability
Available in iOS 4.0 and later.

Declared In
NSFileManager.h

URLsForDirectory:inDomains:

Returns an array of URLs for the specified common directory in the requested domains.

```
- (NSArray *)URLsForDirectory:(NSSearchPathDirectory)directory
    inDomains:(NSSearchPathDomainMask)domainMask
```

Parameters

directory

The search path directory. The supported values are described in `NSSearchPathDirectory` (page 1747).

domainMask

The domain specifies where the search should occur. The constants are specified by `NSSearchPathDomainMask` (page 1750).

Return Value
An array of URLs containing the directories, in the order in which they should be searched.

Discussion
This method is intended to locate known and common directories in the system. For example, setting the directory to `NSApplicationDirectory` (page 1747), will return the Applications directories in the requested domain. There are a number of common directories available in the NSSearchPathDirectory, including: `NSDesktopDirectory` (page 1748), `NSApplicationSupportDirectory` (page 1749), and many more.

Availability
Available in iOS 4.0 and later.

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 *fileManager* is about to attempt to copy.

dstPath

The path or a file or directory to which *fileManager* is about to attempt to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
Returning `NO` from this method causes `NSFileManager` to stop copying the item. If the item skipped is a directory, no children of that directory are copied, nor is the delegate notified of those children.

Availability
Available in iOS 2.0 and later.

See Also
- `copyItemAtPath:toPath:error:` (page 503)
- `fileManager:shouldProceedAfterError:copyingItemAtPath:toPath:` (page 540)
- `fileManager:shouldCopyItemAtURL:toURL:` (page 536)

Declared In
`NSFileManager.h`

fileManager:shouldCopyItemAtURL:toURL:

An `NSFileManager` object sends this message immediately before attempting to copy to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldCopyItemAtURL:(NSURL *)srcURL
    toURL:(NSURL *)dstURL
```

Parameters

fileManager
The `NSFileManager` object that sent this message.

srcURL
The URL or a file or directory that *fileManager* is about to attempt to copy.

dstURL
The URL or a file or directory to which *fileManager* is about to attempt to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
Returning `NO` from this method causes `NSFileManager` to stop copying the item. If the item skipped is a directory, no children of that directory are copied, nor is the delegate notified of those children.

Availability
Available in iOS 4.0 and later.

See Also
- `copyItemAtPath:toPath:error:` (page 503)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 535)
- `fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:` (page 541)

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 *fileManager* is about to attempt to link.

dstPath
The path or a file or directory to which *fileManager* is about to attempt to link.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
Returning NO from this method causes NSFileManager to stop creating the link.

Availability
Available in iOS 2.0 and later.

See Also
- linkItemAtPath:toPath:error: (page 522)
- fileManager:shouldProceedAfterError:linkingItemAtPath:toPath: (page 542)

Declared In
NSFileManager.h

fileManager:shouldLinkItemAtURL:toURL:

An NSFileManager object sends this message immediately before attempting to link to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldLinkItemAtURL:(NSURL *)srcURL
    toURL:(NSURL *)dstURL
```

Parameters
fileManager
The NSFileManager object that sent this message.

srcURL
The URL or a file or directory that *fileManager* is about to attempt to link.

dstURL
The URL or a file or directory to which *fileManager* is about to attempt to link.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
Returning NO from this method causes NSFileManager to stop creating the link.

Availability
Available in iOS 4.0 and later.

See Also
- linkItemAtURL:toURL:error: (page 523)
- fileManager:shouldProceedAfterError:linkingItemAtURL:toURL: (page 542)

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 of a file or directory that *fileManager* is about to attempt to move.

dstPath

The path of a file or directory to which *fileManager* is about to attempt to move.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

Returning `NO` from this method causes `NSFileManager` to stop moving the item. In a move operation, if the source path and the destination path are not on the same device, a copy is performed to the destination path and the original is removed. If the copy does not succeed, `NSFileManager` returns an error and removes the incomplete copy, leaving the original in place.

Availability

Available in iOS 2.0 and later.

See Also

- `moveItemAtPath:toPath:error:` (page 525)
- `fileManager:shouldProceedAfterError:movingItemAtPath:toPath:` (page 543)

Declared In

`NSFileManager.h`

fileManager:shouldMoveItemAtURL:toURL:

An `NSFileManager` object sends this message immediately before attempting to move to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldMoveItemAtURL:(NSURL *)srcURL
    toURL:(NSURL *)dstURL
```

Parameters

fileManager

The `NSFileManager` object that sent this message.

srcURL

The URL of a file or directory that *fileManager* is about to attempt to move.

dstURL

The URL of a file or directory to which *fileManager* is about to attempt to move.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion
Returning `NO` from this method causes `NSFileManager` to stop moving the item. In a move operation, if the source URL and the destination URL are not on the same device, a copy is performed to the destination URL and the original is removed. If the copy does not succeed, `NSFileManager` returns an error and removes the incomplete copy, leaving the original in place.

Availability
Available in iOS 4.0 and later.

See Also
- `moveItemAtURL:toURL:error:` (page 526)
- `fileManager:shouldProceedAfterError:movingItemAtURL:toURL:` (page 544)

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. (Deprecated in iOS 2.0. See delegate methods for copy, move, remove, and link methods.)

```
- (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;
}
```

Special Considerations

The `copyPath:toPath:handler:`, `movePath:toPath:handler:`, `removeFileAtPath:handler:`, and `linkPath:toPath:handler:` methods have all been deprecated as of Mac OS X v10.5. Instead, you can call the `setDelegate:` (page 531) method to specify a delegate that can receive a variety of messages, including messages that replace those described in this section. See the descriptions of the delegate methods in this document for details.

Availability

Available in iOS 2.0 and later.

Deprecated in iOS 2.0.

See Also

- `fileManager:willProcessPath:` (page 547)
- `setDelegate:` (page 531)

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 *fileManager* is attempting to copy.

dstPath

The path or a file or directory to which *fileManager* is attempting to copy.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion
If this method returns `YES`, the `NSFileManager` instance continues as if the error had not occurred. If this method returns `NO`, the `NSFileManager` instance stops copying, and `copyItemAtPath:toPath:error:` (page 503) returns `NO` and provides the error in its `error` argument.

Availability
Available in iOS 2.0 and later.

See Also
- `copyItemAtPath:toPath:error:` (page 503)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 535)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:copyingItemAtURL:toURL:

An `NSFileManager` object sends this message if an error occurs during an attempt to copy to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error copyingItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL
```

Parameters

fileManager
The `NSFileManager` object that sent this message.

error
The error that occurred during the attempt to copy.

srcURL
The URL or a file or directory that *fileManager* is attempting to copy.

dstURL
The URL or a file or directory to which *fileManager* is attempting to copy.

Return Value
`YES` if the operation should proceed, otherwise `NO`.

Discussion
If this method returns `YES`, the `NSFileManager` instance continues as if the error had not occurred. If this method returns `NO`, the `NSFileManager` instance stops copying, and `copyItemAtURL:toURL:error:` (page 504) returns `NO` and provides the error in its `error` argument.

Availability
Available in iOS 4.0 and later.

See Also
- `copyItemAtPath:toPath:error:` (page 503)
- `fileManager:shouldCopyItemAtPath:toPath:` (page 535)

Declared In
`NSFileManager.h`

fileManager:shouldProceedAfterError:linkingItemAtPath:toPath:

An `NSFileManager` object sends this message if an error occurs during an attempt to hard-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 *fileManager* is attempting to link.

dstPath
The path or a file or directory to which *fileManager* is attempting to link.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

If this method returns `YES`, the `NSFileManager` instance continues as if the error had not occurred. If this method returns `NO`, the `NSFileManager` instance stops linking the item, and `linkItemAtPath:toPath:error:` (page 522) returns `NO` and provides the error in its `error` argument.

Availability

Available in iOS 2.0 and later.

See Also

- `linkItemAtPath:toPath:error:` (page 522)
- `fileManager:shouldLinkItemAtPath:toPath:` (page 536)

Declared In

`NSFileManager.h`

fileManager:shouldProceedAfterError:linkingItemAtURL:toURL:

An `NSFileManager` object sends this message if an error occurs during an attempt to hard-link to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error linkingItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL
```

Parameters

fileManager
The `NSFileManager` object that sent this message.

error
The error that occurred during the attempt to link.

srcURL
The URL or a file or directory that *fileManager* is attempting to link.

dstURL
The URL or a file or directory to which *fileManager* is attempting to link.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
If this method returns YES, the NSFileManager instance continues as if the error had not occurred. If this method returns NO, the NSFileManager instance stops linking the item, and linkItemAtURL:toURL:error: (page 523) returns NO and provides the error in its error argument.

Availability
Available in iOS 4.0 and later.

See Also
- linkItemAtURL:toURL:error: (page 523)
- fileManager:shouldLinkItemAtURL:toURL: (page 537)

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 *fileManager* is attempting to move.

dstPath
The path or a file or directory to which *fileManager* is attempting to move.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
If this method returns YES, the NSFileManager instance continues as if the error had not occurred. If this method returns NO, the NSFileManager instance stops moving the item, and moveItemAtPath:toPath:error: (page 525) returns NO and provides the error in its error argument.

Availability
Available in iOS 2.0 and later.

See Also
- moveItemAtPath:toPath:error: (page 525)
- fileManager:shouldMoveItemAtPath:toPath: (page 538)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:movingItemAtURL:toURL:

An NSFileManager object sends this message if an error occurs during an attempt to move to a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error movingItemAtURL:(NSURL *)srcURL toURL:(NSURL *)dstURL
```

Parameters

fileManager
The NSFileManager object that sent this message.

error
The error that occurred during the attempt to move.

srcURL
The URL or a file or directory that *fileManager* is attempting to move.

dstURL
The URL or a file or directory to which *fileManager* is attempting to move.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
If this method returns YES, the NSFileManager instance continues as if the error had not occurred. If this method returns NO, the NSFileManager instance stops moving the item, and moveItemAtURL:toURL:error: (page 526) returns NO and provides the error in its error argument.

Availability
Available in iOS 4.0 and later.

See Also
- moveItemAtURL:toURL:error: (page 526)
- fileManager:shouldMoveItemAtURL:toURL: (page 538)

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 *fileManager* is attempting to delete.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
If this method returns YES, the NSFileManager instance continues as if the error had not occurred. If this method returns NO, the NSFileManager instance stops deleting the item, and removeItemAtPath:error: (page 527) returns NO and provides the error in its error argument.

Availability
Available in iOS 2.0 and later.

See Also
- removeItemAtPath:error: (page 527)
- fileManager:shouldRemoveItemAtPath: (page 546)

Declared In
NSFileManager.h

fileManager:shouldProceedAfterError:removingItemAtURL:

An NSFileManager object sends this message if an error occurs during an attempt to delete a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldProceedAfterError:(NSError
    *)error removingItemAtURL:(NSURL *)URL
```

Parameters
fileManager
The NSFileManager object that sent this message.

error
The error that occurred during the attempt to copy.

URL
The URL or a file or directory that *fileManager* is attempting to delete.

Return Value
YES if the operation should proceed, otherwise NO.

Discussion
If this method returns YES, the NSFileManager instance continues as if the error had not occurred. If this method returns NO, the NSFileManager instance stops deleting the item, and removeItemAtURL:error: (page 528) returns NO and provides the error in its error argument.

Availability
Available in iOS 4.0 and later.

See Also
- removeItemAtURL:error: (page 528)
- fileManager:shouldRemoveItemAtURL: (page 546)

Declared In
NSFileManager.h

fileManager:shouldRemoveItemAtPath:

An `NSFileManager` object sends this message immediately before attempting to delete an item at 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 *`fileManager`* is about to attempt to delete.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

Returning `NO` from this method causes `NSFileManager` to stop deleting the item. If the item is a directory, no children of that item are deleted either.

Availability

Available in iOS 2.0 and later.

See Also

`- removeItemAtPath:error:` (page 527)

`- fileManager:shouldProceedAfterError:removingItemAtPath:` (page 544)

Declared In

`NSFileManager.h`

fileManager:shouldRemoveItemAtURL:

An `NSFileManager` object sends this message immediately before attempting to delete an item at a given URL.

```
- (BOOL)fileManager:(NSFileManager *)fileManager shouldRemoveItemAtURL:(NSURL *)URL
```

Parameters

`fileManager`

The `NSFileManager` object that sent this message.

`URL`

The URL or a file or directory that *`fileManager`* is about to attempt to delete.

Return Value

`YES` if the operation should proceed, otherwise `NO`.

Discussion

Returning `NO` from this method causes `NSFileManager` to stop deleting the item. If the item is a directory, no children of that item are deleted either.

Availability

Available in iOS 4.0 and later.

See Also
- `removeItemAtURL:error:` (page 528)
- `fileManager:shouldProceedAfterError:removingItemAtURL:` (page 545)

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. (Deprecated in iOS 2.0. See delegate methods for copy, move, link, and remove methods.)

```
- (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.

Special Considerations

The `copyPath:toPath:handler:`, `movePath:toPath:handler:`, `removeFileAtPath:handler:`, and `linkPath:toPath:handler:` methods have all been deprecated as of Mac OS X v10.5. Instead, you can call the `setDelegate:` (page 531) method to specify a delegate that can receive a variety of messages, including messages that replace those described in this section. See the descriptions of the delegate methods in this document for details.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 2.0.

Declared In
`NSFileManager.h`

Constants

Mounted Volume Enumeration Options

Options for enumerating mounted volumes with the `mountedVolumeURLsIncludingResourceValuesForKeys:options:` (page 524) method.

```
enum {
     NSVolumeEnumerationSkipHiddenVolumes        = 1L << 1,
     NSVolumeEnumerationProduceFileReferenceURLs = 1L << 2
};
typedef NSUInteger NSVolumeEnumerationOptions;
```

Constants

`NSVolumeEnumerationSkipHiddenVolumes`

The enumeration skips hidden volumes.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

`NSVolumeEnumerationProduceFileReferenceURLs`

The enumeration produces file reference URLs rather than path-based URLs.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

Directory Enumeration Options

Options for enumerating the contents of directories with the `contentsOfDirectoryAtURL:includingPropertiesForKeys:options:error:` (page 502) method.

```
enum {
    NSDirectoryEnumerationSkipsSubdirectoryDescendants = 1L << 0,
    NSDirectoryEnumerationSkipsPackageDescendants      = 1L << 1,
    NSDirectoryEnumerationSkipsHiddenFiles             = 1L << 2
};
typedef NSUInteger NSDirectoryEnumerationOptions;
```

Constants

`NSDirectoryEnumerationSkipsSubdirectoryDescendants`

Perform a shallow enumeration; do not descend into directories.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

`NSDirectoryEnumerationSkipsPackageDescendants`

Do not descend into packages.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

`NSDirectoryEnumerationSkipsHiddenFiles`

Do not enumerate hidden files.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

NSFileManagerItemReplacementOptions

The constants specify the replacement behavior in `NSFileManagerItemReplacementWithoutDeletingBackupItem` (page 549).

```
enum {
    NSFileManagerItemReplacementUsingNewMetadataOnly = 1UL << 0,
    NSFileManagerItemReplacementWithoutDeletingBackupItem = 1UL << 1
};
typedef NSUInteger NSFileManagerItemReplacementOptions;
```

Constants

`NSFileManagerItemReplacementUsingNewMetadataOnly`

Causes `NSFileManagerItemReplacementWithoutDeletingBackupItem` (page 549) to use metadata from the new item only and not to attempt to preserve metadata from the original item.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

`NSFileManagerItemReplacementWithoutDeletingBackupItem`

Causes `NSFileManagerItemReplacementWithoutDeletingBackupItem` (page 549) to leave the backup item in place after a successful replacement. The default behavior is to remove the item.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

File Attribute Keys

These keys access file attribute values contained in `NSDictionary` objects used by `setAttributes:ofItemAtPath:error:` (page 530), `attributesOfItemAtPath:error:` (page 498), `createDirectoryAtPath:withIntermediateDirectories:attributes:error:` (page 506), and `createFileAtPath:contents:attributes:` (page 507).

```
NSString * const NSFileType;
NSString * const NSFileSize;
NSString * const NSFileModificationDate;
NSString * const NSFileReferenceCount;
NSString * const NSFileDeviceIdentifier;
NSString * const NSFileOwnerAccountName;
NSString * const NSFileGroupOwnerAccountName;
NSString * const NSFilePosixPermissions;
NSString * const NSFileSystemNumber;
NSString * const NSFileSystemFileNumber;
NSString * const NSFileExtensionHidden;
NSString * const NSFileHFSCreatorCode;
NSString * const NSFileHFSTypeCode;
NSString * const NSFileImmutable;
NSString * const NSFileAppendOnly;
NSString * const NSFileCreationDate;
NSString * const NSFileOwnerAccountID;
NSString * const NSFileGroupOwnerAccountID;
NSString * const NSFileBusy;
```

```
NSString* const NSFileProtectionKey;
```

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 "NSFileType Attribute Values" (page 552) for possible values).

Available in iOS 2.0 and later.

Declared in `NSFileManager.h`.

`NSFileProtectionKey`

The extended attribute key that identifies the protection level for this file. The corresponding value is an `NSString` value. For a list of possible values, see "File Protection Values" (page 554).

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

Discussion

`NSFileDeviceIdentifier` is used to access the identifier of a remote device.

Declared In

`NSFileManager.h`

NSFileType Attribute Values

These strings are the possible values for the `NSFileType` attribute key contained in the `NSDictionary` object returned by `attributesOfItemAtPath:error:` (page 498).

```
NSString * const NSFileTypeDirectory;
NSString * const NSFileTypeRegular;
NSString * const NSFileTypeSymbolicLink;
NSString * const NSFileTypeSocket;
NSString * const NSFileTypeCharacterSpecial;
NSString * const NSFileTypeBlockSpecial;
NSString * const NSFileTypeUnknown;
```

Constants

`NSFileTypeDirectory`
Directory
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeRegular`
Regular file
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeSymbolicLink`
Symbolic link
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeSocket`
Socket
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeCharacterSpecial`
Character special file
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeBlockSpecial`
Block special file
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

`NSFileTypeUnknown`
Unknown
Available in iOS 2.0 and later.
Declared in `NSFileManager.h`.

File-System Attribute Keys

Keys to access the file attribute values contained in the `NSDictionary` object returned from `NSFileManager`'s `attributesOfFileSystemForPath:error:` (page 497) 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSFileManager.h`.

File Protection Values

Specifies the values that can be associated with the `NSFileProtectionKey` (page 552) key.

```
extern NSString* const NSFileProtectionNone;
extern NSString* const NSFileProtectionComplete;
```

Constants

`NSFileProtectionNone`

The file has no special protections associated with it. It can be read from or written to at any time.

Available in iOS 4.0 and later.

Declared in `NSFileManager.h`.

`NSFileProtectionComplete`

The file is stored in an encrypted format on disk and cannot be read from or written to while the device is locked or booting.

Available in iOS 4.0 and later.

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 iOS 2.0 and later.

Declared in `NSFileManager.h`.

Declared In

`NSFileManager.h`

NSFileWrapper Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSFileWrapper.h
Companion guide	Application File Management

Overview

The `NSFileWrapper` class provides access to the attributes and contents of file-system nodes. A **file-system node** is a file, directory, or symbolic link. Instances of this class are known as **file wrappers**.

File wrappers represent a file-system node as an object that can be displayed as an image (and possibly edited in place), saved to the file system, or transmitted to another application.

There are three types of file wrappers:

- **Regular-file file wrapper:** Represents a regular file.
- **Directory file wrapper:** Represents a directory.
- **Symbolic-link file wrapper:** Represents a symbolic link.

A file wrapper has these attributes:

- **Filename.** Name of the file-system node the file wrapper represents.
- **file-system attributes.** See *NSFileManager Class Reference* for information on the contents of the `attributes` dictionary.
- **Regular-file contents.** Applicable only to regular-file file wrappers.
- **File wrappers.** Applicable only to directory file wrappers.
- **Destination node.** Applicable only to symbolic-link file wrappers.

Adopted Protocols

NSCoding
- `encodeWithCoder:` (page 1552)
- `initWithCoder:` (page 1552)

Tasks

Creating File Wrappers

This class has several designated initializers.

- `initWithURL:options:error:` (page 568)
  Initializes a file wrapper instance whose kind is determined by the type of file-system node located by the URL.
- `initWithPath:` (page 567)
  Initializes a file wrapper instance whose kind is determined by the type of file-system node located by the path. (Deprecated. Use `initWithURL:options:error:` (page 568) instead.)
- `initDirectoryWithFileWrappers:` (page 565)
  Initializes the receiver as a directory file wrapper, with a given file-wrapper list.
- `initRegularFileWithContents:` (page 565)
  Initializes the receiver as a regular-file file wrapper.
- `initSymbolicLinkWithDestinationURL:` (page 567)
  Initializes the receiver as a symbolic-link file wrapper that links to a specified file.
- `initWithSerializedRepresentation:` (page 568)
  Initializes the receiver as a regular-file file wrapper from given serialized data.
- `initSymbolicLinkWithDestination:` (page 566) Deprecated in iOS 4.0
  Initializes the receiver as a symbolic-link file wrapper. (Deprecated. Use `initSymbolicLinkWithDestinationURL:` (page 567) instead.)

Querying File Wrappers

- `isRegularFile` (page 570)
  Indicates whether the receiver is a regular-file file wrapper.
- `isDirectory` (page 569)
  Indicates whether the receiver is a directory file wrapper.
- `isSymbolicLink` (page 570)
  Indicates whether the receiver is a symbolic-link file wrapper.

Accessing File-Wrapper Information

- `fileWrappers` (page 564)
  Returns the file wrappers contained by a directory file wrapper.
- `addFileWrapper:` (page 561)
  Adds a child file wrapper to the receiver, which must be a directory file wrapper.
- `removeFileWrapper:` (page 574)
  Removes a child file wrapper from the receiver, which must be a directory file wrapper.
- `addRegularFileWithContents:preferredFilename:` (page 562)
  Creates a regular-file file wrapper with the given contents and adds it to the receiver, which must be a directory file wrapper.
- `keyForFileWrapper:` (page 570)
  Returns the dictionary key used by a directory to identify a given file wrapper.
- `symbolicLinkDestinationURL` (page 578)
  Provides the URL referenced by the receiver, which must be a symbolic-link file wrapper.
- `addFileWithPath:` (page 560) Deprecated in iOS 4.0
  Creates a file wrapper from a given file-system node and adds it to the receiver, which must be a directory file wrapper. (Deprecated. Use `addFileWrapper:` (page 561) instead.)
- `addSymbolicLinkWithDestination:preferredFilename:` (page 562) Deprecated in iOS 4.0
  Creates a symbolic-link file wrapper pointing to a given file-system node and adds it to the receiver, which must be a directory file wrapper. (Deprecated. Use `addFileWrapper:` (page 561) instead.)
- `symbolicLinkDestination` (page 577) Deprecated in iOS 4.0
  Provides the pathname referenced by the receiver, which must be a symbolic-link file wrapper. (Deprecated. Use `symbolicLinkDestinationURL` (page 578) instead.)

Updating File Wrappers

- `matchesContentsOfURL:` (page 571)
  Indicates whether the contents of a file wrapper matches a directory, regular file, or symbolic link on disk.
- `readFromURL:options:error:` (page 573)
  Recursively rereads the entire contents of a file wrapper from the specified location on disk.
- `needsToBeUpdatedFromPath:` (page 572) Deprecated in iOS 4.0
  Indicates whether the file wrapper needs to be updated to match a given file-system node. (Deprecated. Use `matchesContentsOfURL:` (page 571) instead.)
- `updateFromPath:` (page 578) Deprecated in iOS 4.0
  Updates the file wrapper to match a given file-system node. (Deprecated. Use `readFromURL:options:error:` (page 573) instead.)

Serializing

- `serializedRepresentation` (page 575)
  Returns the contents of the file wrapper as an opaque collection of data.

Accessing Files

- `filename` (page 563)
  Provides the filename of a file wrapper.
- `setFilename:` (page 576)
  Specifies the filename of a file wrapper.
- `preferredFilename` (page 573)
  Provides the preferred filename for a file wrapper.
- `setPreferredFilename:` (page 576)
  Specifies the receiver's preferred filename.
- `fileAttributes` (page 563)
  Returns a file wrapper's file attributes.
- `setFileAttributes:` (page 575)
  Specifies a file wrapper's file attributes.
- `regularFileContents` (page 574)
  Returns the contents of the file-system node associated with a regular-file file wrapper.

Writing Files

- `writeToURL:options:originalContentsURL:error:` (page 579)
  Recursively writes the entire contents of a file wrapper to a given file-system URL.
- `writeToFile:atomically:updateFilenames:` (page 579) Deprecated in iOS 4.0
  Writes a file wrapper's contents to a given file-system node. (Deprecated. Use `writeToURL:options:originalContentsURL:error:` (page 579) instead.)

Instance Methods

addFileWithPath:

Creates a file wrapper from a given file-system node and adds it to the receiver, which must be a directory file wrapper. (Deprecated in iOS 4.0. Use `addFileWrapper:` (page 561) instead.)

```
- (NSString *)addFileWithPath:(NSString *)node
```

Parameters

node
  file-system node from which to create the file wrapper to add to the directory.

Return Value

Dictionary key used to store the new file wrapper in the directory's list of file wrappers. See "Working With Directory Wrappers" for more information.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Instead of using this method, you can instantiate `NSFileWrapper` with one of the initializers, send it `setPreferredFilename:` (page 576) if necessary, and pass the result to `addFileWrapper:` (page 561).

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 4.0.

See Also
- `addRegularFileWithContents:preferredFilename:` (page 562)
- `addSymbolicLinkWithDestination:preferredFilename:` (page 562)
- `removeFileWrapper:` (page 574)
- `fileWrappers` (page 564)

Declared In
`NSFileWrapper.h`

addFileWrapper:

Adds a child file wrapper to the receiver, which must be a directory file wrapper.

```
- (NSString *)addFileWrapper:(NSFileWrapper *)child
```

Parameters

child

File wrapper to add to the directory.

Return Value
Dictionary key used to store *fileWrapper* in the directory's list of file wrappers. The dictionary key is a unique filename, which is the same as the passed-in file wrapper's preferred filename unless that name is already in use as a key in the directory's dictionary of children. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure.

Discussion
Use this method to add an existing file wrapper as a child of a directory file wrapper. If the file wrapper does not have a preferred filename, use the `setPreferredFilename:` (page 576) method to give it one before calling `addFileWrapper:`. To create a new file wrapper and add it to a directory, use the `addRegularFileWithContents:preferredFilename:` (page 562) method.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

This method raises `NSInvalidArgumentException` if the child file wrapper doesn't have a preferred name.

Availability
Available in iOS 4.0 and later.

See Also
- `addRegularFileWithContents:preferredFilename:` (page 562)
- `removeFileWrapper:` (page 574)
- `fileWrappers` (page 564)
- `preferredFilename` (page 573)

Declared In
`NSFileWrapper.h`

addRegularFileWithContents:preferredFilename:

Creates a regular-file file wrapper with the given contents and adds it to the receiver, which must be a directory file wrapper.

```
- (NSString *)addRegularFileWithContents:(NSData *)data preferredFilename:(NSString
    *)filename
```

Parameters

data

Contents for the new regular-file file wrapper.

filename

Preferred filename for the new regular-file file wrapper.

Return Value

Dictionary key used to store the new file wrapper in the directory's list of file wrappers. The dictionary key is a unique filename, which is the same as the passed-in file wrapper's preferred filename unless that name is already in use as a key in the directory's dictionary of children. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure.

Discussion

This is a convenience method. The default implementation allocates a new file wrapper, initializes it with `initRegularFileWithContents:` (page 565), sends it `setPreferredFilename:` (page 576), adds it to the directory with `addFileWrapper:` (page 561), and returns what `addFileWrapper:` returned.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

This method raises `NSInvalidArgumentException` if you pass `nil` or an empty value for `filename`.

Availability

Available in iOS 4.0 and later.

See Also

- `addFileWrapper:` (page 561)
- `removeFileWrapper:` (page 574)
- `fileWrappers` (page 564)

Declared In

`NSFileWrapper.h`

addSymbolicLinkWithDestination:preferredFilename:

Creates a symbolic-link file wrapper pointing to a given file-system node and adds it to the receiver, which must be a directory file wrapper. (Deprecated in iOS 4.0. Use `addFileWrapper:` (page 561) instead.)

```
- (NSString *)addSymbolicLinkWithDestination:(NSString *)node
    preferredFilename:(NSString *)preferredFilename
```

Parameters

node

Pathname the new symbolic-link file wrapper is to reference.

preferredFilename
Preferred filename for the new symbolic-link file wrapper.

Return Value
Dictionary key used to store the new file wrapper in the directory's list of file wrappers. See "Working With Directory Wrappers" for more information.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Instead of using this method, you can instantiate `NSFileWrapper` with one of the initializers, send it `setPreferredFilename:` (page 576) if necessary, and pass the result to `addFileWrapper:` (page 561).

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

This method raises `NSInvalidArgumentException` if you pass `nil` or an empty value for `preferredFilename`.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 4.0.

See Also
- `addFileWrapper:` (page 561)
- `addRegularFileWithContents:preferredFilename:` (page 562)
- `removeFileWrapper:` (page 574)
- `fileWrappers` (page 564)

Declared In
`NSFileWrapper.h`

fileAttributes

Returns a file wrapper's file attributes.

```
- (NSDictionary *)fileAttributes
```

Return Value
File attributes, in a dictionary of the same sort as that returned by `attributesOfItemAtPath:error:` (page 498) (`NSFileManager`).

Availability
Available in iOS 4.0 and later.

See Also
- `setFileAttributes:` (page 575)

Declared In
`NSFileWrapper.h`

filename

Provides the filename of a file wrapper.

```
- (NSString *)filename
```

Return Value
The file wrapper's filename; `nil` when the file wrapper has no corresponding file-system node.

Discussion
The filename is used for record-keeping purposes only and is set automatically when the file wrapper is created from the file system using `initWithURL:options:error:` (page 568) and when it's saved to the file system using `writeToURL:options:originalContentsURL:error:` (page 579) (although this method allows you to request that the filename not be updated).

The filename is usually the same as the preferred filename, but might instead be a name derived from the preferred filename. You can use this method to get the name of a child that's just been read. Don't use this method to get the name of a child that's about to be written, because the name might be about to change; send `keyForFileWrapper:` (page 570) to the parent instead.

Availability
Available in iOS 4.0 and later.

See Also
- `preferredFilename` (page 573)
- `setFilename:` (page 576)

Declared In
`NSFileWrapper.h`

fileWrappers

Returns the file wrappers contained by a directory file wrapper.

```
- (NSDictionary *)fileWrappers
```

Return Value
A key-value dictionary of the file wrappers contained in the directory. The dictionary contains entries whose values are the file wrappers and whose keys are the unique filenames that have been assigned to each one. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure.

Discussion
Returns a dictionary whose values are the file wrapper's children and whose keys are the unique filenames that have been assigned to each one. This method may return `nil` if the user modifies the directory after you call `readFromURL:options:error:` (page 573) or `initWithURL:options:error:` (page 568) but before `NSFileWrapper` has read the contents of the directory. Use the `NSFileWrapperReadingImmediate` reading option to reduce the likelihood of that problem.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

Availability
Available in iOS 4.0 and later.

See Also
- `filename` (page 563)
- `addFileWrapper:` (page 561)

Declared In
NSFileWrapper.h

initDirectoryWithFileWrappers:

Initializes the receiver as a directory file wrapper, with a given file-wrapper list.

```
- (id)initDirectoryWithFileWrappers:(NSDictionary *)childrenByPreferredName
```

Parameters

childrenByPreferredName

Key-value dictionary of file wrappers with which to initialize the receiver. The dictionary must contain entries whose values are the file wrappers that are to become children and whose keys are filenames. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure.

Return Value
Initialized file wrapper for *fileWrappers*.

Discussion
After initialization, the file wrapper is not associated with a file-system node until you save it using `writeToURL:options:originalContentsURL:error:` (page 579).

The receiver is initialized with open permissions: anyone can read, write, or modify the directory on disk.

If any file wrapper in the directory doesn't have a preferred filename, its preferred name is automatically set to its corresponding key in the *childrenByPreferredName* dictionary.

Availability
Available in iOS 4.0 and later.

See Also
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `setFileAttributes:` (page 575)

Declared In
NSFileWrapper.h

initRegularFileWithContents:

Initializes the receiver as a regular-file file wrapper.

```
- (id)initRegularFileWithContents:(NSData *)contents
```

Parameters

contents

Contents of the file.

Return Value
Initialized regular-file file wrapper containing *contents*.

Discussion
After initialization, the file wrapper is not associated with a file-system node until you save it using `writeToURL:options:originalContentsURL:error:` (page 579).

The file wrapper is initialized with open permissions: anyone can write to or read the file wrapper. .

Availability
Available in iOS 4.0 and later.

See Also
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)
- `regularFileContents` (page 574)

Declared In
`NSFileWrapper.h`

initSymbolicLinkWithDestination:

Initializes the receiver as a symbolic-link file wrapper. (Deprecated in iOS 4.0. Use `initSymbolicLinkWithDestinationURL:` (page 567) instead.)

```
- (id)initSymbolicLinkWithDestination:(NSString *)node
```

Parameters

node

Pathname the receiver is to represent.

Return Value
Initialized symbolic-link file wrapper referencing *node*.

Discussion
The receiver is not associated to a file-system node until you save it using `writeToFile:atomically:updateFilenames:` (page 579). It's also initialized with open permissions; anyone can read or write the disk representations it saves.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Therefore, this method has been deprecated in favor of `initSymbolicLinkWithDestinationURL:` (page 567).

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 4.0.

See Also
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)

Declared In
`NSFileWrapper.h`

initSymbolicLinkWithDestinationURL:

Initializes the receiver as a symbolic-link file wrapper that links to a specified file.

```
- (id)initSymbolicLinkWithDestinationURL:(NSURL *)url
```

Parameters

url

URL of the file the file wrapper is to reference.

Return Value

Initialized symbolic-link file wrapper referencing *url*.

Discussion

The file wrapper is not associated with a file-system node until you save it using `writeToURL:options:originalContentsURL:error:` (page 579).

The file wrapper is initialized with open permissions: anyone can modify or read the file reference. .

Availability

Available in iOS 4.0 and later.

See Also

- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)

Declared In

`NSFileWrapper.h`

initWithPath:

Initializes a file wrapper instance whose kind is determined by the type of file-system node located by the path. (Deprecated in iOS 4.0. Use `initWithURL:options:error:` (page 568) instead.)

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

Parameters

node

Pathname of the file-system node the file wrapper is to represent.

Return Value

File wrapper for *node*.

Discussion

If *node* is a directory, this method recursively creates file wrappers for each node within that directory.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Therefore, this method has been deprecated in favor of `initWithURL:options:error:` (page 568).

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 4.0.

See Also
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)

Declared In
`NSFileWrapper.h`

initWithSerializedRepresentation:

Initializes the receiver as a regular-file file wrapper from given serialized data.

```
- (id)initWithSerializedRepresentation:(NSData *)serializedRepresentation
```

Parameters

serializedRepresentation

Serialized representation of a file wrapper in the format used for the `NSFileContentsPboardType` pasteboard type. Data of this format is returned by such methods as `serializedRepresentation` (page 575) and `RTFDFromRange:documentAttributes:` (`NSAttributedString`).

Return Value
Regular-file file wrapper initialized from *serializedRepresentation*.

Discussion
The file wrapper is not associated with a file-system node until you save it using `writeToURL:options:originalContentsURL:error:` (page 579).

Availability
Available in iOS 4.0 and later.

See Also
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)

Declared In
`NSFileWrapper.h`

initWithURL:options:error:

Initializes a file wrapper instance whose kind is determined by the type of file-system node located by the URL.

```
- (id)initWithURL:(NSURL *)url options:(NSFileWrapperReadingOptions)options
    error:(NSError **)outError
```

Parameters

url

URL of the file-system node the file wrapper is to represent.

options
: Option flags for reading the node located at `url`. See "File Wrapper Reading Options" (page 580) for possible values.

outError
: 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
File wrapper for the file-system node at *url*. May be a directory, file, or symbolic link, depending on what is located at the URL. Returns `NO` (0) if reading is not successful.

Discussion
If *url* is a directory, this method recursively creates file wrappers for each node within that directory. Use the `fileWrappers` (page 564) method to get the file wrappers of the nodes contained by the directory.

Availability
Available in iOS 4.0 and later.

See Also
- `fileWrappers` (page 564)
- `setPreferredFilename:` (page 576)
- `filename` (page 563)
- `fileAttributes` (page 563)
- `readFromURL:options:error:` (page 573)

Declared In
`NSFileWrapper.h`

isDirectory

Indicates whether the receiver is a directory file wrapper.

```
- (BOOL)isDirectory
```

Return Value
`YES` when the receiver is a directory file wrapper, `NO` otherwise.

Discussion
Invocations of `readFromURL:options:error:` (page 573) may change what is returned by subsequent invocations of this method if the type of the file on disk has changed.

Availability
Available in iOS 4.0 and later.

See Also
- `isRegularFile` (page 570)
- `isSymbolicLink` (page 570)

Declared In
`NSFileWrapper.h`

isRegularFile

Indicates whether the receiver is a regular-file file wrapper.

```
- (BOOL)isRegularFile
```

Return Value
`YES` when the receiver is a regular-file wrapper, `NO` otherwise.

Discussion
Invocations of `readFromURL:options:error:` (page 573) may change what is returned by subsequent invocations of this method if the type of the file on disk has changed.

Availability
Available in iOS 4.0 and later.

See Also
- `isDirectory` (page 569)
- `isSymbolicLink` (page 570)

Declared In
`NSFileWrapper.h`

isSymbolicLink

Indicates whether the receiver is a symbolic-link file wrapper.

```
- (BOOL)isSymbolicLink
```

Return Value
`YES` when the receiver is a symbolic-link file wrapper, `NO` otherwise.

Discussion
Invocations of `readFromURL:options:error:` (page 573) may change what is returned by subsequent invocations of this method if the type of the file on disk has changed.

Availability
Available in iOS 4.0 and later.

See Also
- `isDirectory` (page 569)
- `isRegularFile` (page 570)

Declared In
`NSFileWrapper.h`

keyForFileWrapper:

Returns the dictionary key used by a directory to identify a given file wrapper.

```
- (NSString *)keyForFileWrapper:(NSFileWrapper *)child
```

Parameters

child

The child file wrapper for which you want the key.

Return Value

Dictionary key used to store the file wrapper in the directory's list of file wrappers. The dictionary key is a unique filename, which may not be the same as the passed-in file wrapper's preferred filename if more than one file wrapper in the directory's dictionary of children has the same preferred filename. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure. Returns `nil` if the file wrapper specified in `child` is not a child of the directory.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

Availability

Available in iOS 4.0 and later.

See Also

`- filename` (page 563)

Declared In

`NSFileWrapper.h`

matchesContentsOfURL:

Indicates whether the contents of a file wrapper matches a directory, regular file, or symbolic link on disk.

```
- (BOOL)matchesContentsOfURL:(NSURL *)url
```

Parameters

url

URL of the file-system node with which to compare the file wrapper.

Return Value

`YES` when the contents of the file wrapper match the contents of *url*, `NO` otherwise.

Discussion

The contents of files are not compared; matching of regular files is based on file modification dates. For a directory, children are compared against the files in the directory, recursively.

Because children of directory file wrappers are not read immediately by the `initWithURL:options:error:` (page 568) method unless the `NSFileWrapperReadingImmediate` reading option is used, even a newly-created directory file wrapper might not have the same contents as the directory on disk. You can use this method to determine whether the file wrapper's contents in memory need to be updated.

If the file wrapper needs updating, use the `readFromURL:options:error:` (page 573) method with the `NSFileWrapperReadingImmediate` reading option.

This table describes which attributes of the file wrapper and file-system node are compared to determine whether the file wrapper matches the node on disk:

File-wrapper type	Comparison determinants
Regular file	Modification date and access permissions.
Directory	Children (recursive).
Symbolic link	Destination pathname.

Availability
Available in iOS 4.0 and later.

See Also
- `readFromURL:options:error:` (page 573)
- `fileAttributes` (page 563)

Declared In
`NSFileWrapper.h`

needsToBeUpdatedFromPath:

Indicates whether the file wrapper needs to be updated to match a given file-system node. (Deprecated in iOS 4.0. Use `matchesContentsOfURL:` (page 571) instead.)

```
- (BOOL)needsToBeUpdatedFromPath:(NSString *)node
```

Parameters

node

file-system node with which to compare the file wrapper.

Return Value
`YES` when the file wrapper needs to be updated to match *node*, `NO` otherwise.

Discussion
This table describes which attributes of the file wrapper and *node* are compared to determine whether the file wrapper needs to be updated:

File-wrapper type	Comparison determinants
Regular file	Modification date and access permissions.
Directory	Member hierarchy (recursive).
Symbolic link	Destination pathname.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Therefore, this method has been deprecated in favor of `matchesContentsOfURL:` (page 571).

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 4.0.

See Also
- `updateFromPath:` (page 578)
- `fileAttributes` (page 563)

Declared In
`NSFileWrapper.h`

preferredFilename

Provides the preferred filename for a file wrapper.

```
- (NSString *)preferredFilename
```

Return Value
The file wrapper's preferred filename.

Discussion
This name is normally used as the dictionary key when a child file wrapper is added to a directory file wrapper. However, if another file wrapper with the same preferred name already exists in the directory file wrapper when the receiver is added, the filename assigned as the dictionary key may differ from the preferred filename.

Availability
Available in iOS 4.0 and later.

See Also
- `filename` (page 563)
- `setPreferredFilename:` (page 576)

Declared In
`NSFileWrapper.h`

readFromURL:options:error:

Recursively rereads the entire contents of a file wrapper from the specified location on disk.

```
- (BOOL)readFromURL:(NSURL *)url options:(NSFileWrapperReadingOptions)options
    error:(NSError **)outError
```

Parameters

url
URL of the file-system node corresponding to the file wrapper.

options
Option flags for reading the node located at `url`. See "File Wrapper Reading Options" (page 580) for possible values.

outError
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 successful. If not successful, returns `NO` after setting `outError` to an `NSError` object that describes the reason why the file wrapper could not be reread.

Discussion
When reading a directory, children are added and removed as necessary to match the file system.

Availability
Available in iOS 4.0 and later.

See Also
- `initWithURL:options:error:` (page 568)
- `fileWrappers` (page 564)
- `filename` (page 563)
- `fileAttributes` (page 563)
- `writeToURL:options:originalContentsURL:error:` (page 579)

Declared In
`NSFileWrapper.h`

regularFileContents

Returns the contents of the file-system node associated with a regular-file file wrapper.

```
- (NSData *)regularFileContents
```

Return Value
Contents of the file-system node the file wrapper represents.

Discussion
This method may return `nil` if the user modifies the file after you call `readFromURL:options:error:` (page 573) or `initWithURL:options:error:` (page 568) but before `NSFileWrapper` has read the contents of the file. Use the `NSFileWrapperReadingImmediate` reading option to reduce the likelihood of that problem.

Special Considerations
This method raises `NSInternalInconsistencyException` if the receiver is not a regular-file file wrapper.

Availability
Available in iOS 4.0 and later.

See Also
- `initRegularFileWithContents:` (page 565)
- `readFromURL:options:error:` (page 573)

Declared In
`NSFileWrapper.h`

removeFileWrapper:

Removes a child file wrapper from the receiver, which must be a directory file wrapper.

```
- (void)removeFileWrapper:(NSFileWrapper *)child
```

Parameters
child
File wrapper to remove from the directory.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a directory file wrapper.

Availability

Available in iOS 4.0 and later.

See Also

- `addFileWrapper:` (page 561)
- `addRegularFileWithContents:preferredFilename:` (page 562)
- `fileWrappers` (page 564)

Declared In

`NSFileWrapper.h`

serializedRepresentation

Returns the contents of the file wrapper as an opaque collection of data.

```
- (NSData *)serializedRepresentation
```

Return Value

The file wrapper's contents in the format used for the pasteboard type `NSFileContentsPboardType`.

Discussion

Returns an `NSData` object suitable for passing to `initWithSerializedRepresentation:` (page 568). This method may return `nil` if the user modifies the contents of the file-system node after you call `readFromURL:options:error:` (page 573) or `initWithURL:options:error:` (page 568) but before `NSFileWrapper` has read the contents of the file. Use the `NSFileWrapperReadingImmediate` reading option to reduce the likelihood of that problem.

Availability

Available in iOS 4.0 and later.

See Also

- `initWithSerializedRepresentation:` (page 568)

Declared In

`NSFileWrapper.h`

setFileAttributes:

Specifies a file wrapper's file attributes.

```
- (void)setFileAttributes:(NSDictionary *)fileAttributes
```

Parameters

fileAttributes

File attributes for the file wrapper, in a dictionary of the same sort as that used by `setAttributes:ofItemAtPath:error:` (page 530) (`NSFileManager`).

Availability

Available in iOS 4.0 and later.

See Also
- `fileAttributes` (page 563)
- `writeToURL:options:originalContentsURL:error:` (page 579)

Declared In
`NSFileWrapper.h`

setFilename:

Specifies the filename of a file wrapper.

```
- (void)setFilename:(NSString *)filename
```

Parameters

filename

Filename of the file wrapper.

Discussion

The file name is a dictionary key used to store *fileWrapper* in a directory's list of child file wrappers. The dictionary key is a unique filename, which is the same as the child file wrapper's preferred filename unless that name is already in use as a key in the directory's dictionary of children. See "Working With Directory Wrappers" in *Application File Management* for more information about the file-wrapper list structure. In general, the filename is set for you by the `initWithURL:options:error:` (page 568), `initDirectoryWithFileWrappers:` (page 565), or `writeToURL:options:originalContentsURL:error:` (page 579) methods; you do not normally have to call this method directly.

Special Considerations

This method raises `NSInvalidArgumentException` if you pass `nil` or an empty value for `filename`.

Availability

Available in iOS 4.0 and later.

See Also
- `filename` (page 563)
- `setPreferredFilename:` (page 576)
- `initWithURL:options:error:` (page 568)
- `initDirectoryWithFileWrappers:` (page 565)
- `writeToURL:options:originalContentsURL:error:` (page 579)

Declared In
`NSFileWrapper.h`

setPreferredFilename:

Specifies the receiver's preferred filename.

```
- (void)setPreferredFilename:(NSString *)filename
```

Parameters

filename

Preferred filename for the receiver.

Discussion

When a file wrapper is added to a parent directory file wrapper, the parent attempts to use the child's preferred filename as the key in its dictionary of children. If that key is already in use, then the parent derives a unique filename from the preferred filename and uses that for the key.

When you change the preferred filename of a file wrapper, the default implementation of this method causes existing parent directory file wrappers to remove and re-add the child to accommodate the change. Preferred filenames of children are not preserved when you write a file wrapper to disk and then later instantiate another file wrapper by reading the file from disk. If you need to preserve the user-visible names of attachments, you have to store the names yourself.

Special Considerations

This method raises `NSInvalidArgumentException` if you pass `nil` or an empty value for `filename`.

Availability

Available in iOS 4.0 and later.

See Also

- `preferredFilename` (page 573)
- `setFilename:` (page 576)
- `addFileWrapper:` (page 561)
- `initWithURL:options:error:` (page 568)
- `initDirectoryWithFileWrappers:` (page 565)
- `writeToURL:options:originalContentsURL:error:` (page 579)

Declared In

`NSFileWrapper.h`

symbolicLinkDestination

Provides the pathname referenced by the receiver, which must be a symbolic-link file wrapper. (Deprecated in iOS 4.0. Use `symbolicLinkDestinationURL` (page 578) instead.)

```
- (NSString *)symbolicLinkDestination
```

Return Value

Pathname the file wrapper references (the destination of the symbolic link the file wrapper represents).

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Therefore, this method has been deprecated in favor of `symbolicLinkDestinationURL` (page 578).

This method raises `NSInternalInconsistencyException` if the receiver is not a symbolic-link file wrapper.

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 4.0.

Declared In
NSFileWrapper.h

symbolicLinkDestinationURL

Provides the URL referenced by the receiver, which must be a symbolic-link file wrapper.

```
- (NSURL *)symbolicLinkDestinationURL
```

Return Value
Pathname the file wrapper references (that is, the destination of the symbolic link the file wrapper represents).

Discussion
This method may return `nil` if the user modifies the symbolic link after you call `readFromURL:options:error:` (page 573) or `initWithURL:options:error:` (page 568) but before `NSFileWrapper` has read the contents of the link. Use the `NSFileWrapperReadingImmediate` reading option to reduce the likelihood of that problem.

Special Considerations

This method raises `NSInternalInconsistencyException` if the receiver is not a symbolic-link file wrapper.

Availability
Available in iOS 4.0 and later.

Declared In
NSFileWrapper.h

updateFromPath:

Updates the file wrapper to match a given file-system node. (Deprecated in iOS 4.0. Use `readFromURL:options:error:` (page 573) instead.)

```
- (BOOL)updateFromPath:(NSString *)path
```

Return Value
`YES` if the update is carried out, `NO` if it isn't needed.

Discussion
For a directory file wrapper, the contained file wrappers are also sent `updateFromPath:` messages. If nodes in the corresponding directory on the file system have been added or removed, corresponding file wrappers are released or created as needed.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a `file://` URL. Therefore, this method has been deprecated in favor of `readFromURL:options:error:` (page 573).

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 4.0.

See Also
`- needsToBeUpdatedFromPath:` (page 572)

Declared In
NSFileWrapper.h

writeToFile:atomically:updateFilenames:

Writes a file wrapper's contents to a given file-system node. (Deprecated in iOS 4.0. Use writeToURL:options:originalContentsURL:error: (page 579) instead.)

```
- (BOOL)writeToFile:(NSString *)node atomically:(BOOL)atomically
    updateFilenames:(BOOL)updateNames
```

Parameters

node

Pathname of the file-system node to which the receiver's contents are written.

atomically

YES to write the file safely so that:

- An existing file is not overwritten
- The method fails if the file cannot be written in its entirety

NO to overwrite an existing file and ignore incomplete writes.

updateNames

YES to update the receiver's filenames (its filename and—for directory file wrappers—the filenames of its sub–file wrappers) be changed to the filenames of the corresponding nodes in the file system, after a successful write operation. Use this in Save or Save As operations.

NO to specify that the receiver's filenames not be updated. Use this in Save To operations.

Return Value
YES when the write operation is successful, NO otherwise.

Special Considerations

Beginning with Mac OS X v10.6, the preferred method of referring to files is with a file:// URL. Therefore, this method has been deprecated in favor of writeToURL:options:originalContentsURL:error: (page 579).

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 4.0.

See Also
- filename (page 563)
- writeToURL:options:originalContentsURL:error: (page 579)

Declared In
NSFileWrapper.h

writeToURL:options:originalContentsURL:error:

Recursively writes the entire contents of a file wrapper to a given file-system URL.

```
- (BOOL)writeToURL:(NSURL *)url options:(NSFileWrapperWritingOptions)options
    originalContentsURL:(NSURL *)originalContentsURL error:(NSError **)outError
```

Parameters

url

URL of the file-system node to which the file wrapper's contents are written.

options

Option flags for writing to the node located at `url`. See "File Wrapper Writing Options" (page 581) for possible values.

originalContentsURL

The location of a previous revision of the contents being written. The default implementation of this method attempts to avoid unnecessary I/O by writing hard links to regular files instead of actually writing out their contents when the contents have not changed. The child file wrappers must return accurate values when sent the `filename` (page 563) method for this to work. Use the `NSFileWrapperWritingWithNameUpdating` writing option to increase the likelihood of that.

Specify `nil` for this parameter if there is no earlier version of the contents or if you want to ensure that all the contents are written to files.

updateNames

`YES` to update the receiver's filenames (its filename and—for directory file wrappers—the filenames of its sub–file wrappers) be changed to the filenames of the corresponding nodes in the file system, after a successful write operation. Use this in Save or Save As operations.

`NO` to specify that the receiver's filenames not be updated. Use this in Save To operations.

outError

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` when the write operation is successful. If not successful, returns `NO` after setting `outError` to an `NSError` object that describes the reason why the file wrapper's contents could not be written.

Availability

Available in iOS 4.0 and later.

See Also

- `filename` (page 563)
- `readFromURL:options:error:` (page 573)

Declared In

`NSFileWrapper.h`

Constants

File Wrapper Reading Options

Reading options that can be set by the `initWithURL:options:error:` (page 568) and `readFromURL:options:error:` (page 573) methods.

```
enum {
 NSFileWrapperReadingImmediate = 1 << 0,
 NSFileWrapperReadingWithoutMapping = 1 << 1
};
typedef NSUInteger NSFileWrapperReadingOptions;
```

Constants

`NSFileWrapperReadingImmediate`

If reading with this option succeeds, then subsequent invocations of `fileWrappers` (page 564), `regularFileContents` (page 574), `symbolicLinkDestinationURL` (page 578), and `serializedRepresentation` (page 575) sent to the file wrapper and all its child file wrappers will fail and return `nil` only if an actual error occurs (for example, the volume has disappeared or the file server is unreachable)—not as a result of the user moving or deleting files.

For performance reasons, `NSFileWrapper` may not read the contents of some file packages immediately even when this option is chosen. For example, because the contents of bundles (not all file packages are bundles) are immutable to the user, `NSFileWrapper` may read the children of such a directory lazily.

You can use this option to take a snapshot of a file or folder for writing later. For example, an application like TextEdit can use this option when creating new file wrappers to represent attachments that the user creates by copying and pasting or dragging and dropping from the Finder to a TextEdit document. Don't use this option when reading a document file package, because that would cause unnecessarily bad performance. For example, an application wouldn't use this option when creating file wrappers to represent attachments as it's opening a document stored in a file package.

Available in iOS 4.0 and later.

Declared in `NSFileWrapper.h`.

`NSFileWrapperReadingWithoutMapping`

Whether file mapping for regular file wrappers is disallowed.

You can use this option to keep `NSFileWrapper` from memory-mapping files. This is useful if you want to make sure your application doesn't hold files open (mapped files are open files), therefore preventing the user from ejecting DVDs, unmounting disk partitions, or unmounting disk images. In Mac OS X v10.6 and later, `NSFileWrapper` memory-maps files that are on internal drives only. It never memory-maps files on external drives or network volumes, regardless of whether this option is used.

Available in iOS 4.0 and later.

Declared in `NSFileWrapper.h`.

Discussion

You can use the `NSFileWrapperReadingImmediate` and `NSFileWrapperReadingWithoutMapping` reading options together to take an exact snapshot of a file-system hierarchy that is safe from all errors (including the ones mentioned above) once reading has succeeded. If reading with both options succeeds, then subsequent invocations of the methods listed in the comment for the `NSFileWrapperReadingImmediate` reading option to the receiver and all its descendant file wrappers will never fail. However, note that reading with both options together is expensive in terms of both I/O and memory for large files, or directories containing large files, or even directories containing many small files.

File Wrapper Writing Options

Writing options that can be set by the `writeToURL:options:originalContentsURL:error:` (page 579) method.

```
enum {
 NSFileWrapperWritingAtomic = 1 << 0,
 NSFileWrapperWritingWithNameUpdating = 1 << 1
};
typedef NSUInteger NSFileWrapperWritingOptions;
```

Constants

`NSFileWrapperWritingAtomic`

Whether writing is done atomically.

You can use this option to ensure that, when overwriting a file package, the overwriting either completely succeeds or completely fails, with no possibility of leaving the file package in an inconsistent state. Because this option causes additional I/O, you shouldn't use it unnecessarily. For example, don't use this option in an override of `-[NSDocument writeToURL:ofType:error:]`, because `NSDocument` safe-saving is already done atomically.

Available in iOS 4.0 and later.

Declared in `NSFileWrapper.h`.

`NSFileWrapperWritingWithNameUpdating`

Whether descendant file wrappers are sent the `setFilename:` (page 576) method if the writing succeeds.

This option is necessary when your application passes a URL in the `originalContentsURL` parameter to the `writeToURL:options:originalContentsURL:error:` (page 579) method. Without using this option (and reusing child file wrappers properly), subsequent invocations of `writeToURL:options:originalContentsURL:error:` (page 579) would not be able to reliably create hard links in a new file package, because the record of names in the old file package would be out of date.

Available in iOS 4.0 and later.

Declared in `NSFileWrapper.h`.

NSFormatter Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSFormatter.h
Companion guide	Data Formatting Guide

Overview

`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 588)
The default implementation of this method raises an exception.

- attributedStringForObjectValue:withDefaultAttributes: (page 584)
  The default implementation returns nil to indicate that the formatter object does not provide an attributed string.
- editingStringForObjectValue: (page 585)
  The default implementation of this method invokes stringForObjectValue: (page 588).

Object Equivalent to Textual Representation

- getObjectValue:forString:errorDescription: (page 585)
  The default implementation of this method raises an exception.

Dynamic Cell Editing

- isPartialStringValid:newEditingString:errorDescription: (page 587)
  Returns a Boolean value that indicates whether a partial string is valid.
- isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription: (page 587)
  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 588) to get the non-attributed string, then create an NSAttributedString object with it (see initWithString: (page 98)). 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 String Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
`- editingStringForObjectValue:` (page 585)

Declared In
`NSFormatter.h`

editingStringForObjectValue:

The default implementation of this method invokes `stringForObjectValue:` (page 588).

```
- (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 588). As an example, you would implement this method if you want the dollar signs in displayed strings removed for editing.

Availability
Available in iOS 2.0 and later.

See Also
`- attributedStringForObjectValue:withDefaultAttributes:` (page 584)

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 588)) 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;
}
```

Special Considerations

Prior to Mac OS X v10.6, the implementation of this method in both NSNumberFormatter and NSDateFormatter would return YES and an object value even if only part of the string could be parsed. This is problematic because you cannot be sure what portion of the string was parsed. For applications linked on or after Mac OS X v10.6, this method instead returns an error if part of the string cannot be parsed. You can use getObjectValue:forString:range:error: to get the old behavior—it returns the range of the substring that was successfully parsed.

Availability
Available in iOS 2.0 and later.

See Also
- stringForObjectValue: (page 588)

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 587), this method will be called as before (`isPartialStringValid:proposedSelectedRange:originalString:originalSelectedRange:errorDescription:` (page 587) just calls this one by default).

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`- isPartialStringValid:newEditingString:errorDescription:` (page 587)

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 585) 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 585) 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 iOS 2.0 and later.

See Also

- `attributedStringForObjectValue:withDefaultAttributes:` (page 584)
- `editingStringForObjectValue:` (page 585)
- `getObjectValue:forString:errorDescription:` (page 585)

Declared In

`NSFormatter.h`

NSHTTPCookie Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSHTTPCookie.h
Companion guide	URL Loading System Programming Guide

Overview

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 1554)

Tasks

Create Cookie Instances

`+ cookiesWithResponseHeaderFields:forURL:` (page 593)
Returns an array of NSHTTPCookie objects corresponding to the provided response header fields for the provided URL.

+ `cookieWithProperties:` (page 593)
  Creates and initializes an NSHTTPCookie object using the provided properties.
- `initWithProperties:` (page 595)
  Returns an initialized `NSHTTPCookie` object using the provided properties.

Convert Cookies to Request Headers

+ `requestHeaderFieldsWithCookies:` (page 594)
  Returns a dictionary of header fields corresponding to a provided array of cookies.

Getting Cookie Properties

- `comment` (page 594)
  Returns the receiver's comment string.
- `commentURL` (page 594)
  Returns the receiver's comment URL.
- `domain` (page 595)
  Returns the domain of the receiver's cookie.
- `expiresDate` (page 595)
  Returns the receiver's expiration date.
- `isHTTPOnly` (page 596)
  Returns whether the receiver should only be sent to HTTP servers per RFC 2965.
- `isSecure` (page 596)
  Returns whether his cookie should only be sent over secure channels.
- `isSessionOnly` (page 597)
  Returns whether the receiver should be discarded at the end of the session (regardless of expiration date).
- `name` (page 597)
  Returns the receiver's name.
- `path` (page 597)
  Returns the receiver's path.
- `portList` (page 598)
  Returns the receiver's port list.
- `properties` (page 598)
  Returns the receiver's cookie properties.
- `value` (page 598)
  Returns the receiver's value.
- `version` (page 599)
  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 created cookies.

Discussion
This method ignores irrelevant header fields in *headerFields*, allowing dictionaries to contain additional data.

If *headerFields* does not specify a domain for a given cookie, the cookie is created with a default domain value of *theURL*.

If *headerFields* does not specify a path for a given cookie, the cookie is created with a default path value of "/".

Availability

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 599) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability

See Also
- `initWithProperties:` (page 595)

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 599) for details on the header field keys and values in the returned dictionary.

Availability

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

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

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 is only sent to the exact host specified by the domain. If the domain does start with a dot, then the cookie is sent to other hosts in that domain as well, subject to certain restrictions. See RFC 2965 for more detail.

Availability

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

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 599) for more information on the available header field constants and the constraints imposed on the values in the dictionary.

Availability

See Also

`+ cookieWithProperties:` (page 593)

Declared In

`NSHTTPCookie.h`

isHTTPOnly

Returns whether the receiver should only be sent to HTTP servers per RFC 2965.

```
- (BOOL)isHTTPOnly
```

Return Value

Returns `YES` if this cookie should only be sent via HTTP headers, `NO` otherwise.

Discussion

Cookies may be marked as HTTP only by a server (or by a javascript). Cookies marked as such must only be sent via HTTP Headers in HTTP requests for URL's that match both the path and domain of the respective cookies.

Important: Cookies specified as HTTP only should not be delivered to any javascript applications to prevent cross-site scripting vulnerabilities.

Availability

Available in iOS 2.2 and later.

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

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

Declared In
NSHTTPCookie.h

name

Returns the receiver's name.

```
- (NSString *)name
```

Return Value
The receiver's name.

Availability

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

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

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 595) or `cookieWithProperties:` (page 593) to create an equivalent `NSHTTPCookie` object.

See `initWithProperties:` (page 595) for more information on the constraints imposed on the *`properties`* dictionary.

Availability

Declared In
`NSHTTPCookie.h`

value

Returns the receiver's value.

```
- (NSString *)value
```

Return Value
The receiver's value.

Availability

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

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieName`

An `NSString` object containing the name of the cookie. This field is required.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookiePath`

An `NSString` object containing the path for the cookie. This field is required if you are using the `NSHTTPCookieDomain` key instead of the `NSHTTPCookieOriginURL` key.

If you are using the `NSHTTPCookieOriginURL` key, the path is inferred if it is not provided. The default value is "/".

Available in iOS 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 iOS 2.0 and later.

Declared in `NSHTTPCookie.h`.

`NSHTTPCookieSecure`

An `NSString` object indicating that the cookie should be transmitted only over secure channels.

Providing any value for this key indicates that the cookie should remain secure.

Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSHTTPCookieStorage.h
Companion guide	URL Loading System Programming Guide

Overview

`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.

iOS Note: Cookies are not shared among applications in iOS.

Note: Changes made to the cookie accept policy affect all currently running applications using the cookie storage.

Tasks

Getting the Shared Cookie Storage Object

`+ sharedHTTPCookieStorage` (page 604)
Returns the shared cookie storage instance.

Getting and Setting the Cookie Accept Policy

`- cookieAcceptPolicy` (page 604)
Returns the cookie storage's cookie accept policy.

`- setCookieAcceptPolicy:` (page 606)
Sets the cookie accept policy of the cookie storage.

Adding and Removing Cookies

- `cookies` (page 605)
  Returns the cookie storage's cookies.
- `cookiesForURL:` (page 605)
  Returns all the cookie storage's cookies that are sent to a specified URL.
- `deleteCookie:` (page 605)
  Deletes the specified cookie from the cookie storage.
- `setCookie:` (page 606)
  Stores a specified cookie in the cookie storage if the cookie accept policy permits.
- `setCookies:forURL:mainDocumentURL:` (page 607)
  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

Declared In
`NSHTTPCookieStorage.h`

Instance Methods

cookieAcceptPolicy

Returns the cookie storage's cookie accept policy.

```
- (NSHTTPCookieAcceptPolicy)cookieAcceptPolicy
```

Return Value
The cookie storage's cookie accept policy. The default cookie accept policy is `NSHTTPCookieAcceptPolicyAlways`.

Availability

See Also
- `setCookieAcceptPolicy:` (page 606)

Declared In
NSHTTPCookieStorage.h

cookies

Returns the cookie storage's cookies.

```
- (NSArray *)cookies
```

Return Value
An array containing all of the cookie storage's cookies.

Availability

See Also
- cookiesForURL: (page 605)

Declared In
NSHTTPCookieStorage.h

cookiesForURL:

Returns all the cookie storage's cookies that are 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 method requestHeaderFieldsWithCookies: (page 594) to turn this array into a set of header fields to add to an NSMutableURLRequest object.

Availability

See Also
- cookies (page 605)

Declared In
NSHTTPCookieStorage.h

deleteCookie:

Deletes the specified cookie from the cookie storage.

```
- (void)deleteCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to delete.

Availability

Declared In
NSHTTPCookieStorage.h

setCookie:

Stores a specified cookie in the cookie storage if the cookie accept policy permits.

```
- (void)setCookie:(NSHTTPCookie *)aCookie
```

Parameters

aCookie

The cookie to store.

Discussion
The cookie replaces an existing cookie with the same name, domain, and path, if one exists in the cookie storage. This method accepts the cookie only if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyAlways` or `NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`. The cookie is ignored if the receiver's cookie accept policy is `NSHTTPCookieAcceptPolicyNever`.

Availability

Declared In
NSHTTPCookieStorage.h

setCookieAcceptPolicy:

Sets the cookie accept policy of the cookie storage.

```
- (void)setCookieAcceptPolicy:(NSHTTPCookieAcceptPolicy)aPolicy
```

Parameters

aPolicy

The new cookie accept policy.

Discussion
The default cookie accept policy is `NSHTTPCookieAcceptPolicyAlways`. Changing the cookie policy affects all currently running applications using the cookie storage.

Availability

See Also
`- cookieAcceptPolicy` (page 604)

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 593) 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

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 iOS 2.0 and later.

Declared in `NSHTTPCookieStorage.h`.

`NSHTTPCookieAcceptPolicyNever`
Reject all cookies.

Available in iOS 2.0 and later.

Declared in `NSHTTPCookieStorage.h`.

`NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`
Accept cookies only from the main document domain.

Available in iOS 2.0 and later.

Declared in `NSHTTPCookieStorage.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSHTTPCookieStorage.h`

Notifications

NSHTTPCookieManagerCookiesChangedNotification

This notification is posted when the cookies stored in the `NSHTTPCookieStorage` instance have changed.

In Mac OS X, cookies are shared among applications, meaning this notification can be sent in response to another application's actions. Cookies are not shared among applications in iOS.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability

Declared In
`NSHTTPCookieStorage.h`

NSHTTPCookieManagerAcceptPolicyChangedNotification

This notification is posted when the acceptance policy of the `NSHTTPCookieStorage` instance has changed.

In Mac OS X, cookies are shared among applications, meaning this notification can be sent in response to another application's actions. Cookies are not shared among applications in iOS.

The notification object is the `NSHTTPCookieStorage` instance. This notification does not contain a `userInfo` dictionary.

Availability

Declared In
`NSHTTPCookieStorage.h`

NSHTTPURLResponse Class Reference

Inherits from	NSURLResponse : NSObject
Conforms to	NSCoding (NSURLResponse) NSCopying (NSURLResponse) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLResponse.h
Companion guide	URL Loading System Programming Guide

Overview

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 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Getting HTTP Response Headers

`- allHeaderFields` (page 610)
Returns all the HTTP header fields of the receiver.

Getting Response Status Code

`+ localizedStringForStatusCode:` (page 610)
Returns a localized string corresponding to a specified HTTP status code.

`- statusCode` (page 611)
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

See Also
`- statusCode` (page 611)

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

Declared In
NSURLResponse.h

statusCode

Returns the receiver's HTTP status code.

```
- (NSInteger)statusCode
```

Return Value
The receiver's HTTP status code.

Availability

See Also
`+ localizedStringForStatusCode:` (page 610)

Declared In
`NSURLResponse.h`

NSIndexPath Class Reference

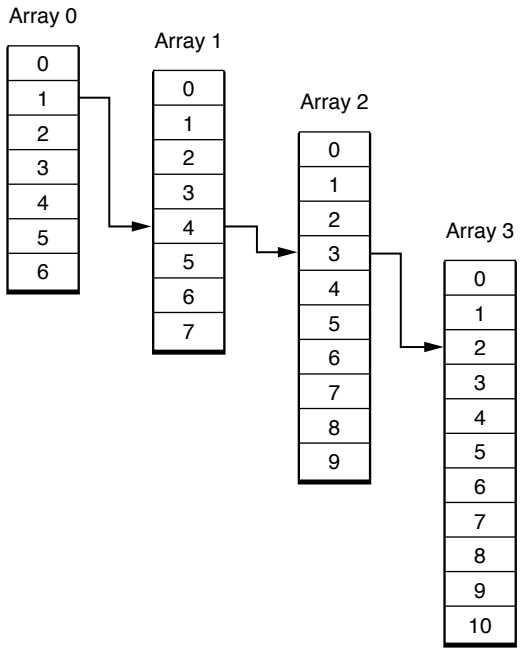
Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSIndexPath.h
Companion guide	Collections Programming Topics
Related sample code	BonjourWeb CryptoExercise GKRocket MultipleDetailViews WiTap

Overview

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 37-1.

Figure 37-1 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 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Creating Index Paths

`+ indexPathWithIndex:` (page 615)
Creates an one-node index path.

`+ indexPathWithIndexes:length:` (page 616)
Creates an index path with one or more nodes.

`- initWithIndex:` (page 618)
Initializes an allocated `NSIndexPath` (page 613) object with a one-node index path.

- `initWithIndexes:length:` (page 619)
  Initializes an allocated `NSIndexPath` (page 613) object with an index path of a specific length.

Querying Index Paths

- `getIndexes:` (page 617)
  Provides a reference to the receiver's indexes.
- `indexAtPosition:` (page 617)
  Provides the index at a particular node in the receiver.
- `indexPathByAddingIndex:` (page 617)
  Provides an index path containing the indexes in the receiver and another index.
- `indexPathByRemovingLastIndex` (page 618)
  Provides an index path with the indexes in the receiver, excluding the last one.
- `length` (page 619)
  Provides the number of indexes in the receiver.

Comparing Index Paths

- `compare:` (page 616)
  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 iOS 2.0 and later.

See Also
- `initWithIndex:` (page 618)

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 iOS 2.0 and later.

See Also
`- initWithIndexes:length:` (page 619)

Related Sample Code
BonjourWeb

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 iOS 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 iOS 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
Index value of the desired node. Node numbering starts at zero.

Return Value
Index value at *node*.

Availability
Available in iOS 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 613) object containing the receiver's indexes and *index*.

Availability
Available in iOS 2.0 and later.

See Also
- `indexPathByRemovingLastIndex` (page 618)

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.

Discussion
Returns an empty `NSIndexPath` instance if the receiver's length is 1 or less.

Special Considerations

On Mac OS X 10.4 and earlier this method returns `nil` when the length of the receiver is 1 or less. On Mac OS X 10.5 and later this method will never return `nil`.

Availability
Available in iOS 2.0 and later.

See Also
- `indexPathByAddingIndex:` (page 617)

Declared In
`NSIndexPath.h`

initWithIndex:

Initializes an allocated `NSIndexPath` (page 613) 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 613) object representing a one-node index path with *index*.

Availability
Available in iOS 2.0 and later.

See Also
+ `indexPathWithIndex:` (page 615)

Declared In
`NSIndexPath.h`

initWithIndexes:length:

Initializes an allocated `NSIndexPath` (page 613) 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 613) object with *indexes* up to *length*.

Availability
Available in iOS 2.0 and later.

See Also
`+ indexPathWithIndexes:length:` (page 616)

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 iOS 2.0 and later.

Declared In
`NSIndexPath.h`

NSIndexSet Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSIndexSet.h
Companion guide	Collections Programming Topics
Related sample code	BonjourWeb

Overview

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 637) and `initWithIndexSet:` (page 637).

You must not subclass the `NSIndexSet` class.

The mutable subclass of `NSIndexSet` is NSMutableIndexSet.

Adopted Protocols

NSCoding
- `- encodeWithCoder:` (page 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

NSMutableCopying
- `- mutableCopyWithZone:` (page 1614)

Tasks

Creating Index Sets

`+ indexSet` (page 624)
Creates an empty index set.

`+ indexSetWithIndex:` (page 624)
Creates an index set with an index.

`+ indexSetWithIndexesInRange:` (page 625)
Creates an index set with an index range.

`- init` (page 636)
Initializes an allocated `NSIndexSet` (page 621) object.

`- initWithIndex:` (page 637)
Initializes an allocated `NSIndexSet` (page 621) object with an index.

`- initWithIndexesInRange:` (page 637)
Initializes an allocated `NSIndexSet` (page 621) object with an index range.

`- initWithIndexSet:` (page 637)
Initializes an allocated `NSIndexSet` (page 621) object with an index set.

Querying Index Sets

`- containsIndex:` (page 625)
Indicates whether the receiver contains a specific index.

`- containsIndexes:` (page 626)
Indicates whether the receiver contains a superset of the indexes in another index set.

`- containsIndexesInRange:` (page 626)
Indicates whether the receiver contains the indexes represented by an index range.

`- intersectsIndexesInRange:` (page 638)
Indicates whether the receiver contains any of the indexes in a range.

- `count` (page 627)
  Returns the number of indexes in the receiver.
- `countOfIndexesInRange:` (page 627)
  Returns the number of indexes in the receiver that are members of a given range.
- `indexPassingTest:` (page 635)
  Returns the index of the first object that passes the predicate Block test.
- `indexesPassingTest:` (page 631)
  Returns an `NSIndexSet` containing the receiver's objects that pass the Block test.
- `indexWithOptions:passingTest:` (page 636)
  Returns the index of the first object that passes the predicate Block test using the specified enumeration options.
- `indexesWithOptions:passingTest:` (page 632)
  Returns an `NSIndexSet` containing the receiver's objects that pass the Block test using the specified enumeration options.
- `indexInRange:options:passingTest:` (page 633)
  Returns the index of the first object in the specified range that passes the predicate Block test.
- `indexesInRange:options:passingTest:` (page 630)
  Returns an `NSIndexSet` containing the receiver's objects in the specified range that pass the Block test.

Comparing Index Sets

- `isEqualToIndexSet:` (page 638)
  Indicates whether the indexes in the receiver are the same indeces contained in another index set.

Getting Indexes

- `firstIndex` (page 629)
  Returns either the first index in the receiver or the not-found indicator.
- `lastIndex` (page 639)
  Returns either the last index in the receiver or the not-found indicator.
- `indexLessThanIndex:` (page 634)
  Returns either the closest index in the receiver that is less than a specific index or the not-found indicator.
- `indexLessThanOrEqualToIndex:` (page 634)
  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 633)
  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 632)
  Returns either the closest index in the receiver that is greater than a specific index or the not-found indicator.

- `getIndexes:maxCount:inIndexRange:` (page 629)
  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.

Enumerating Indexes

- `enumerateIndexesUsingBlock:` (page 628)
  Executes a given Block using each object in the receiver.
- `enumerateIndexesWithOptions:usingBlock:` (page 628)
  Executes a given Block over the receiver's indexes, using the specified enumeration options.
- `enumerateIndexesInRange:options:usingBlock:` (page 627)
  Executes a given Block using the indexes in the specified range, using the specified enumeration options.

Class Methods

indexSet

Creates an empty index set.

```
+ (id)indexSet
```

Return Value
`NSIndexSet` (page 621) object with no members.

Availability
Available in iOS 2.0 and later.

See Also
`- init` (page 636)

Declared In
`NSIndexSet.h`

indexSetWithIndex:

Creates an index set with an index.

```
+ (id)indexSetWithIndex:(NSUInteger)index
```

Parameters
index
An index.

Return Value
`NSIndexSet` (page 621) object containing *index*.

Availability
Available in iOS 2.0 and later.

See Also
`- initWithIndex:` (page 637)

Related Sample Code
BonjourWeb

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 621) object containing *indexRange*.

Availability
Available in iOS 2.0 and later.

See Also
`- initWithIndexesInRange:` (page 637)

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 iOS 2.0 and later.

See Also
- `containsIndexes:` (page 626)
- `containsIndexesInRange:` (page 626)

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 iOS 2.0 and later.

See Also
- `containsIndex:` (page 625)
- `containsIndexesInRange:` (page 626)

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 iOS 2.0 and later.

See Also
- `containsIndex:` (page 625)
- `containsIndexes:` (page 626)
- `intersectsIndexesInRange:` (page 638)

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 iOS 2.0 and later.

See Also
- countOfIndexesInRange: (page 627)

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 iOS 2.0 and later.

See Also
- count (page 627)

Declared In
NSIndexSet.h

enumerateIndexesInRange:options:usingBlock:

Executes a given Block using the indexes in the specified range, using the specified enumeration options.

```
- (void)enumerateIndexesInRange:(NSRange)range options:(NSEnumerationOptions)opts
    usingBlock:(void (^)(NSUInteger idx, BOOL *stop))block
```

Parameters
range
Index to enumerate.

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See `NSEnumerationOptions` (page 1745) for the supported values.

block

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

Availability
Available in iOS 4.0 and later.

Declared In
`NSIndexSet.h`

enumerateIndexesUsingBlock:

Executes a given Block using each object in the receiver.

```
- (void)enumerateIndexesUsingBlock:(void (^)(NSUInteger idx, BOOL *stop))block
```

Parameters

block

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

Availability
Available in iOS 4.0 and later.

Declared In
`NSIndexSet.h`

enumerateIndexesWithOptions:usingBlock:

Executes a given Block over the receiver's indexes, using the specified enumeration options.

```
- (void)enumerateIndexesWithOptions:(NSEnumerationOptions)opts usingBlock:(void
    (^)(NSUInteger idx, BOOL *stop))block
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See `NSEnumerationOptions` (page 1745) for the supported values.

block

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

Availability
Available in iOS 4.0 and later.

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 1757) when the receiver is empty.

Availability
Available in iOS 2.0 and later.

See Also
`- lastIndex` (page 639)

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 iOS 2.0 and later.

Declared In

`NSIndexSet.h`

indexesInRange:options:passingTest:

Returns an `NSIndexSet` containing the receiver's objects in the specified range that pass the Block test.

```
- (NSIndexSet *)indexesInRange:(NSRange)range options:(NSEnumerationOptions)opts
    passingTest:(BOOL (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

range

The range of indexes to test.

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See `NSEnumerationOptions` (page 1745) for the supported values.

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
An `NSIndexSet` containing the indexes of the receiver that passed the predicate Block test.

Availability
Available in iOS 4.0 and later.

Declared In
`NSIndexSet.h`

indexesPassingTest:

Returns an `NSIndexSet` containing the receiver's objects that pass the Block test.

```
- (NSIndexSet *)indexesPassingTest:(BOOL (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
An `NSIndexSet` containing the indexes of the receiver that passed the predicate Block test.

Availability
Available in iOS 4.0 and later.

Declared In
`NSIndexSet.h`

indexesWithOptions:passingTest:

Returns an `NSIndexSet` containing the receiver's objects that pass the Block test using the specified enumeration options.

```
- (NSIndexSet *)indexesWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL
    (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See `NSEnumerationOptions` (page 1745) for the supported values.

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
An `NSIndexSet` containing the indexes of the receiver that passed the predicate Block test.

Availability
Available in iOS 4.0 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 1757) when the receiver contains no qualifying index.

Availability
Available in iOS 2.0 and later.

See Also
`- indexLessThanIndex:` (page 634)

- indexGreaterThanOrEqualToIndex: (page 633)
- indexLessThanOrEqualToIndex: (page 634)

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 1757) when the receiver contains no qualifying index.

Availability
Available in iOS 2.0 and later.

See Also
- indexGreaterThanIndex: (page 632)
- indexLessThanIndex: (page 634)
- indexLessThanOrEqualToIndex: (page 634)

Declared In
NSIndexSet.h

indexInRange:options:passingTest:

Returns the index of the first object in the specified range that passes the predicate Block test.

```
- (NSUInteger)indexInRange:(NSRange)range options:(NSEnumerationOptions)opts
    passingTest:(BOOL (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

range

The range of indexes to test.

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See NSEnumerationOptions (page 1745) for the supported values.

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The index of the first object that passes the predicate test.

Availability
Available in iOS 4.0 and later.

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 1757) when the receiver contains no qualifying index.

Availability
Available in iOS 2.0 and later.

See Also
- `indexGreaterThanIndex:` (page 632)
- `indexGreaterThanOrEqualToIndex:` (page 633)
- `indexLessThanOrEqualToIndex:` (page 634)

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 1757) when the receiver contains no qualifying index.

Availability

Available in iOS 2.0 and later.

See Also

- `indexGreaterThanIndex:` (page 632)
- `indexLessThanIndex:` (page 634)
- `indexGreaterThanOrEqualToIndex:` (page 633)

Declared In

`NSIndexSet.h`

indexPassingTest:

Returns the index of the first object that passes the predicate Block test.

```
- (NSUInteger)indexPassingTest:(BOOL (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value

The index of the first object that passes the predicate test.

Availability

Available in iOS 4.0 and later.

Declared In

`NSIndexSet.h`

indexWithOptions:passingTest:

Returns the index of the first object that passes the predicate Block test using the specified enumeration options.

```
- (NSUInteger)indexWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL
    (^)(NSUInteger idx, BOOL *stop))predicate
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order). See `NSEnumerationOptions` (page 1745) for the supported values.

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

idx

The index of the object.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
The index of the first object that passes the predicate test.

Availability
Available in iOS 4.0 and later.

Declared In
`NSIndexSet.h`

init

Initializes an allocated `NSIndexSet` (page 621) object.

```
- (id)init
```

Return Value
Initialized, empty `NSIndexSet` (page 621) object.

Availability
Available in iOS 2.0 and later.

See Also
`+ indexSet` (page 624)

Declared In
`NSIndexSet.h`

initWithIndex:

Initializes an allocated NSIndexSet (page 621) object with an index.

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

Parameters

index

An index.

Return Value

Initialized NSIndexSet (page 621) object with *index*.

Availability

Available in iOS 2.0 and later.

See Also

+ indexSetWithIndex: (page 624)

Declared In

NSIndexSet.h

initWithIndexesInRange:

Initializes an allocated NSIndexSet (page 621) 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 621) object with *indexRange*.

Discussion

This method raises an NSRangeException (page 1773) when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

This method is a designated initializer for NSIndexSet (page 621).

Availability

Available in iOS 2.0 and later.

See Also

+ indexSetWithIndexesInRange: (page 625)

Declared In

NSIndexSet.h

initWithIndexSet:

Initializes an allocated NSIndexSet (page 621) object with an index set.

```
- (id)initWithIndexSet:(NSIndexSet *)indexSet
```

Parameters

indexSet

An index set.

Return Value

Initialized `NSIndexSet` (page 621) object with *indexSet*.

Discussion

This method is a designated initializer for `NSIndexSet` (page 621).

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`- containsIndexesInRange:` (page 626)

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 iOS 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 1757) when the receiver is empty.

Availability
Available in iOS 2.0 and later.

See Also
- firstIndex (page 629)

Declared In
NSIndexSet.h

NSInputStream Class Reference

Inherits from	NSStream : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CryptoExercise WiTap

Overview

`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 re-implement existing initializers. You must also provide complete implementations of the following methods:

- `read:maxLength:` (page 646)

  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 644)

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 645)

  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 643)
Creates and returns an initialized `NSInputStream` object for reading from a given `NSData` object.

`+ inputStreamWithFileAtPath:` (page 643)
Creates and returns an initialized `NSInputStream` object that reads data from the file at a given path.

`+ inputStreamWithURL:` (page 644)
Creates and returns an initialized `NSInputStream` object that reads data from the file at a given URL.

`- initWithData:` (page 645)
Initializes and returns an `NSInputStream` object for reading from a given `NSData` object.

`- initWithFileAtPath:` (page 645)
Initializes and returns an `NSInputStream` object that reads data from the file at a given path.

`- initWithURL:` (page 646)
Initializes and returns an `NSInputStream` object that reads data from the file at a given URL.

Using Streams

`- read:maxLength:` (page 646)
Reads up to a given number of bytes into a given buffer.

`- getBuffer:length:` (page 644)
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 645)
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 iOS 2.0 and later.

See Also

`+ inputStreamWithFileAtPath:` (page 643)

`- initWithData:` (page 645)

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 iOS 2.0 and later.

See Also

`+ inputStreamWithData:` (page 643)

`- initWithFileAtPath:` (page 645)

`- initWithURL:` (page 646)

Declared In

`NSStream.h`

inputStreamWithURL:

Creates and returns an initialized `NSInputStream` object that reads data from the file at a given URL.

```
+ (id)inputStreamWithURL:(NSURL *)url
```

Parameters

url

The URL to the file.

Return Value

An initialized `NSInputStream` object that reads data from the URL at *url*. If the file specified by *url* doesn't exist or is unreadable, returns `nil`.

Availability

Available in iOS 4.0 and later.

See Also

`+ inputStreamWithData:` (page 643)

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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- initWithFileAtPath:` (page 645)
`+ inputStreamWithData:` (page 643)

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 iOS 2.0 and later.

See Also
- `initWithData:` (page 645)
+ `inputStreamWithFileAtPath:` (page 643)
+ `inputStreamWithURL:` (page 644)

Declared In
`NSStream.h`

initWithURL:

Initializes and returns an `NSInputStream` object that reads data from the file at a given URL.

```
- (id)initWithURL:(NSURL *)url
```

Parameters
url
The URL to the file.

Return Value
An initialized `NSInputStream` object that reads data from the file at *url*. If the file specified by *url* doesn't exist or is unreadable, returns `nil`.

Availability
Available in iOS 4.0 and later.

See Also
- `initWithData:` (page 645)

Declared In
`NSStream.h`

read:maxLength:

Reads up to a given number of bytes into a given buffer.

```
- (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
A number indicating the outcome of the operation:

- A positive number indicates the number of bytes read;
- `0` indicates that the end of the buffer was reached;
- A negative number means that the operation failed.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSInvocation.h
Companion guide	Distributed Objects Programming Topics

Overview

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 651) class method to create `NSInvocation` objects; you should not create these objects using `alloc` (page 949) and `init` (page 971).

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 1552)
- `initWithCoder:` (page 1552)

Tasks

Creating NSInvocation Objects

`+ invocationWithMethodSignature:` (page 651)
Returns an `NSInvocation` object able to construct messages using a given method signature.

Configuring an Invocation Object

`- setSelector:` (page 656)
Sets the receiver's selector.

`- selector` (page 655)
Returns the receiver's selector, or 0 if it hasn't been set.

`- setTarget:` (page 657)
Sets the receiver's target.

`- target` (page 657)
Returns the receiver's target, or `nil` if the receiver has no target.

`- setArgument:atIndex:` (page 655)
Sets an argument of the receiver.

`- getArgument:atIndex:` (page 652)
Returns by indirection the receiver's argument at a specified index.

`- argumentsRetained` (page 651)
Returns `YES` if the receiver has retained its arguments, `NO` otherwise.

`- retainArguments` (page 654)
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 656)
Sets the receiver's return value.

`- getReturnValue:` (page 652)
Gets the receiver's return value.

Dispatching an Invocation

- `invoke` (page 653)
  Sends the receiver's message (with arguments) to its target and sets the return value.
- `invokeWithTarget:` (page 654)
  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 654)
  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 656) and its arguments set with `setArgument:atIndex:` (page 655) before it can be invoked. Do not use the `alloc` (page 949)/`init` (page 971) approach to create `NSInvocation` objects.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `retainArguments` (page 654)

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 iOS 2.0 and later.

See Also
- `setArgument:atIndex:` (page 655)
- `numberOfArguments` (page 725) (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 724) to determine the size needed for *buffer*:

```
NSUInteger 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 iOS 2.0 and later.

See Also

- `setReturnValue:` (page 656)
- `methodReturnType` (page 725) (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 iOS 2.0 and later.

See Also

- `getReturnValue:` (page 652)
- `setSelector:` (page 656)
- `setTarget:` (page 657)
- `setArgument:atIndex:` (page 655)

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 iOS 2.0 and later.

See Also

- `getReturnValue:` (page 652)
- `invoke` (page 653)
- `setSelector:` (page 656)
- `setTarget:` (page 657)
- `setArgument:atIndex:` (page 655)

Declared In

`NSInvocation.h`

methodSignature

Returns the receiver's method signature.

```
- (NSMethodSignature *)methodSignature
```

Availability

Available in iOS 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 651) 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 iOS 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 iOS 2.0 and later.

See Also
- setSelector: (page 656)

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 657) and setSelector: (page 656) 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 iOS 2.0 and later.

See Also
- `getArgument:atIndex:` (page 652)
- `numberOfArguments` (page 725) (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 653) or `invokeWithTarget:` (page 654) message.

Availability
Available in iOS 2.0 and later.

See Also
- `getReturnValue:` (page 652)
- `methodReturnLength` (page 724) (NSMethodSignature)
- `methodReturnType` (page 725) (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 iOS 2.0 and later.

See Also
- `selector` (page 655)

Declared In
NSInvocation.h

setTarget:

Sets the receiver's target.

```
- (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 653).

Discussion

Availability
Available in iOS 2.0 and later.

See Also
- `target` (page 657)
- `invokeWithTarget:` (page 654)

Declared In
NSInvocation.h

target

Returns the receiver's target, or `nil` if the receiver has no target.

```
- (id)target
```

Availability
Available in iOS 2.0 and later.

See Also
- `setTarget:` (page 657)

Declared In
NSInvocation.h

NSInvocationOperation Class Reference

Inherits from	NSOperation : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Threading Programming Guide
Related sample code	CryptoExercise

Overview

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 660)
Returns an `NSInvocationOperation` object initialized with the specified target and selector.

`- initWithInvocation:` (page 660)
Returns an `NSInvocationOperation` object initialized with the specified invocation object.

Getting Attributes

`- invocation` (page 661)
Returns the receiver's invocation object.

- `result` (page 661)
  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 iOS 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 must be `id`. The return type of the method may be `void`, a scalar value, or an object that can be returned as an `id` type.

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.

Discussion
If you specify a selector with a non-void return type, you can get the return value by calling the `result` (page 661) method after the operation finishes executing. The receiver tells the invocation object to retain its arguments.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`- initWithTarget:selector:object:` (page 660)
`- initWithInvocation:` (page 660)

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 662).

Availability
Available in iOS 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 661) 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide

Overview

`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 672)
  Returns the receiver, initialized for encoding an archive into a given a mutable-data object.

Archiving Data

+ `archivedDataWithRootObject:` (page 665)
  Returns an `NSData` object containing the encoded form of the object graph whose root object is given.

+ `archiveRootObject:toFile:` (page 666)
  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 672)
  Instructs the receiver to construct the final data stream.

- `outputFormat` (page 672)
  Returns the format in which the receiver encodes its data.

- `setOutputFormat:` (page 674)
  Sets the format in which the receiver encodes its data.

Encoding Data and Objects

- `encodeBool:forKey:` (page 668)
  Encodes a given Boolean value and associates it with a given key.

- `encodeBytes:length:forKey:` (page 668)
  Encodes a given number of bytes from a given C array of bytes and associates them with the a given key.

- `encodeConditionalObject:forKey:` (page 669)
  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 671).

- `encodeDouble:forKey:` (page 669)
  Encodes a given `double` value and associates it with a given key.

- `encodeFloat:forKey:` (page 670)
  Encodes a given `float` value and associates it with a given key.

- `encodeInt:forKey:` (page 671)
  Encodes a given `int` value and associates it with a given key.

- `encodeInt32:forKey:` (page 670)
  Encodes a given 32-bit integer value and associates it with a given key.

- `encodeInt64:forKey:` (page 670)
  Encodes a given 64-bit integer value and associates it with a given key.

- `encodeObject:forKey:` (page 671)
  Encodes a given object and associates it with a given key.

Managing Delegates

- `delegate` (page 668)
  Returns the receiver's delegate.
- `setDelegate:` (page 673)
  Sets the delegate for the receiver.

Managing Classes and Class Names

+ `setClassName:forClass:` (page 667)
  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 666)
  Returns the class name with which `NSKeyedArchiver` encodes instances of a given class.
- `setClassName:forClass:` (page 673)
  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 667)
  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 iOS 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 iOS 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 iOS 2.0 and later.

See Also

`+ setClassName:forClass:` (page 667)

`- classNameForClass:` (page 667)

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 iOS 2.0 and later.

See Also

`+ classNameForClass:` (page 666)

`- setClassName:forClass:` (page 673)

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 iOS 2.0 and later.

See Also

`- setClassName:forClass:` (page 673)

`+ classNameForClass:` (page 666)

Declared In

`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id < NSKeyedArchiverDelegate >)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 673)

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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeBoolForKey:` (page 680) (`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. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeBytesForKey:returnedLength:` (page 680) (`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 671).

```
- (void)encodeConditionalObject:(id)objv forKey:(NSString *)key
```

Parameters
objv
The object to encode.

key
The key with which to associate the encoded value. This value must not be `nil`.

Availability
Available in iOS 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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeDoubleForKey:` (page 681) (`NSKeyedUnarchiver`)
`decodeFloatForKey:` (page 681) (`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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeFloatForKey:` (page 681) (`NSKeyedUnarchiver`)
`decodeDoubleForKey:` (page 681) (`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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeInt32ForKey:` (page 682) (`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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeInt64ForKey:` (page 682) (`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*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeIntForKey:` (page 683) (`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. This value may be `nil`.

key
The key with which to associate *objv*. This value must not be `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`decodeObjectForKey:` (page 683) (`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 iOS 2.0 and later.

See Also
`- initForWritingWithMutableData:` (page 672)

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 672) at which point *data* is filled. The format of the receiver is `NSPropertyListBinaryFormat_v1_0`.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `setOutputFormat:` (page 674)

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 iOS 2.0 and later.

See Also
- `classNameForClass:` (page 667)
+ `setClassName:forClass:` (page 667)

Declared In
`NSKeyedArchiver.h`

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id < NSKeyedArchiverDelegate >)delegate
```

Parameters

delegate

The delegate for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `delegate` (page 668)

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 iOS 2.0 and later.

See Also
`- outputFormat` (page 672)

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 iOS 2.0 and later.

Declared in `NSKeyedArchiver.h`.

Declared In
`NSKeyedArchiver.h`

NSKeyedUnarchiver Class Reference

Inherits from	NSCoder : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide

Overview

`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 680) returns `NO`, `decodeIntForKey:` (page 683) returns `0`, and `decodeObjectForKey:` (page 683) 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 684)
  Initializes the receiver for decoding an archive previously encoded by `NSKeyedArchiver`.

Unarchiving Data

+ `unarchiveObjectWithData:` (page 678)
  Decodes and returns the object graph previously encoded by `NSKeyedArchiver` and stored in a given `NSData` object.
+ `unarchiveObjectWithFile:` (page 679)
  Decodes and returns the object graph previously encoded by `NSKeyedArchiver` written to the file at a given path.

Decoding Data

- `containsValueForKey:` (page 680)
  Returns a Boolean value that indicates whether the archive contains a value for a given key within the current decoding scope.
- `decodeBoolForKey:` (page 680)
  Decodes a Boolean value associated with a given key.
- `decodeBytesForKey:returnedLength:` (page 680)
  Decodes a stream of bytes associated with a given key.
- `decodeDoubleForKey:` (page 681)
  Decodes a double-precision floating-point value associated with a given key.
- `decodeFloatForKey:` (page 681)
  Decodes a single-precision floating-point value associated with a given key.
- `decodeIntForKey:` (page 683)
  Decodes an integer value associated with a given key.
- `decodeInt32ForKey:` (page 682)
  Decodes a 32-bit integer value associated with a given key.
- `decodeInt64ForKey:` (page 682)
  Decodes a 64-bit integer value associated with a given key.
- `decodeObjectForKey:` (page 683)
  Decodes and returns an object associated with a given key.
- `finishDecoding` (page 684)
  Tells the receiver that you are finished decoding objects.

Managing the Delegate

- `delegate` (page 684)
  Returns the receiver's delegate.
- `setDelegate:` (page 685)
  Sets the receiver's delegate.

Managing Class Names

+ `setClass:forClassName:` (page 678)
  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 677)
  Returns the class from which `NSKeyedUnarchiver` instantiates an encoded object with a given class name.
- `setClass:forClassName:` (page 685)
  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 679)
  Returns the class from which the receiver instantiates an encoded object with a given class name.

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 iOS 2.0 and later.

See Also
+ `setClass:forClassName:` (page 678)
- `classForClassName:` (page 679)

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 iOS 2.0 and later.

See Also
`+ classForClassName:` (page 677)
`- setClass:forClassName:` (page 685)

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 674) if *data* is not a valid archive.

Availability
Available in iOS 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` (page 1773) if the file at *path* does not contain a valid archive.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`- setClass:forClassName:` (page 685)

`+ classForClassName:` (page 677)

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. *key* must not be `nil`.

Return Value

`YES` if the archive contains a value for *key* within the current decoding scope, otherwise `NO`.

Availability

Available in iOS 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. *key* must not be `nil`.

Return Value

The Boolean value associated with the key *key*. Returns `NO` if *key* does not exist.

Availability

Available in iOS 2.0 and later.

See Also

`encodeBool:forKey:` (page 668) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeBytes:length:forKey:` (page 668) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeDouble:forKey:` (page 669) (`NSKeyedArchiver`)
`encodeFloat:forKey:` (page 670) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeFloat:forKey:` (page 670) (`NSKeyedArchiver`)
`encodeDouble:forKey:` (page 669) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeInt32:forKey:` (page 670) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeInt64:forKey:` (page 670) (`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. *key* must not be `nil`.

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 iOS 2.0 and later.

See Also
`encodeInt:forKey:` (page 671) (`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. *key* must not be `nil`.

Return Value
The object associated with the key *key*. Returns `nil` if *key* does not exist, or if the value for *key* is `nil`.

Availability
Available in iOS 2.0 and later.

See Also
`encodeObject:forKey:` (page 671) (`NSKeyedArchiver`)

Declared In
`NSKeyedArchiver.h`

delegate

Returns the receiver's delegate.

```
- (id < NSKeyedUnarchiverDelegate >)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 685)

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 iOS 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 684).

This method raises an `NSInvalidArchiveOperationException` (page 674) if *`data`* is not a valid archive.

Availability
Available in iOS 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 678)).

Availability
Available in iOS 2.0 and later.

See Also
`- classForClassName:` (page 679)
`+ setClass:forClassName:` (page 678)

Declared In
`NSKeyedArchiver.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id < NSKeyedUnarchiverDelegate >)delegate
```

Parameters

`delegate`
The delegate for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- delegate (page 684)

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.

```
NSString *NSInvalidUnarchiveOperationException;
```

Constants

NSInvalidUnarchiveOperationException

The name of the exception raised by NSKeyedArchiver if there is a problem extracting an archive.

Available in iOS 2.0 and later.

Declared in NSKeyedArchiver.h.

NSLocale Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSLocale.h
Companion guides	Locales Programming Guide Data Formatting Guide

Overview

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 Guide*). 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 698)
Initializes the receiver using a given locale identifier.

+ systemLocale (page 696)
Returns the "root", canonical locale, that contains fixed "backstop" settings that provide values for otherwise undefined keys.

+ currentLocale (page 692)
Returns the logical locale for the current user.

+ autoupdatingCurrentLocale (page 689)
Returns the current logical locale for the current user.

Getting Information About a Locale

- displayNameForKey:value: (page 697)
Returns the display name for the given value.

- localeIdentifier (page 698)
Returns the identifier for the receiver.

- objectForKey: (page 699)
Returns the object corresponding to the specified key.

Getting System Locale Information

+ availableLocaleIdentifiers (page 690)
Returns an array of NSString objects, each of which identifies a locale available on the system.

+ ISOCountryCodes (page 693)
Returns an array of NSString objects that represents all known legal country codes.

+ ISOCurrencyCodes (page 693)
Returns an array of NSString objects that represents all known legal ISO currency codes.

+ ISOLanguageCodes (page 694)
Returns an array of NSString objects that represents all known legal ISO language codes.

+ commonISOCurrencyCodes (page 691)
Returns an array of common ISO currency codes

Converting Between Identifiers

+ canonicalLocaleIdentifierFromString: (page 690)
Returns the canonical identifier for a given locale identification string.

+ componentsFromLocaleIdentifier: (page 692)
Returns a dictionary that is the result of parsing a locale ID.

+ localeIdentifierFromComponents: (page 695)
Returns a locale identifier from the components specified in a given dictionary.

+ canonicalLanguageIdentifierFromString: (page 690)
Returns a canonical language identifier by mapping an arbitrary locale identification string to the canonical identifier.

+ localeIdentifierFromWindowsLocaleCode: (page 695)
Returns a locale identifier from a Windows locale code.

`+ windowsLocaleCodeFromLocaleIdentifier:` (page 696)
Returns a Window locale code from the locale identifier.

Getting Preferred Languages

`+ preferredLanguages` (page 696)
Returns the user's language preference order as an array of strings.

Getting Line and Character Direction For a Language

`+ characterDirectionForLanguage:` (page 691)
Returns the character direction for the specified ISO language code.

`+ lineDirectionForLanguage:` (page 694)
Returns the line direction for the specified ISO language code.

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 692)).

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* to learn how to register for and receive notifications).

Availability
Available in iOS 2.0 and later.

See Also
`+ systemLocale` (page 696)
`+ currentLocale` (page 692)

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 iOS 2.0 and later.

See Also
`+ ISOLanguageCodes` (page 694)
`+ ISOCountryCodes` (page 693)
`+ ISOCurrencyCodes` (page 693)
`+ commonISOCurrencyCodes` (page 691)

Declared In
`NSLocale.h`

canonicalLanguageIdentifierFromString:

Returns a canonical language identifier by mapping an arbitrary locale identification string to the canonical identifier.

```
+ (NSString *)canonicalLanguageIdentifierFromString:(NSString *)string
```

Parameters
string
A string representation of an arbitrary locale identifier.

Return Value
A string that represents the canonical language identifier for the specified arbitrary locale identifier.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
+ componentsFromLocaleIdentifier: (page 692)
+ localeIdentifierFromComponents: (page 695)

Declared In
NSLocale.h

characterDirectionForLanguage:

Returns the character direction for the specified ISO language code.

```
+ (NSLocaleLanguageDirection)characterDirectionForLanguage:(NSString *)isoLangCode
```

Parameters

isoLangCode
The ISO language code.

Return Value
Returns the character direction for the language. See "NSLocaleLanguageDirection" (page 699) for possible values. If the appropriate direction can't be determined NSLocaleLanguageDirectionUnknown (page 699) is returned.

Availability
Available in iOS 4.0 and later.

See Also
+ lineDirectionForLanguage: (page 694)

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 iOS 2.0 and later.

See Also
+ availableLocaleIdentifiers (page 690)
+ ISOCountryCodes (page 693)

+ ISOCurrencyCodes (page 693)

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 699).

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 iOS 2.0 and later.

See Also
+ localeIdentifierFromComponents: (page 695)
+ canonicalLocaleIdentifierFromString: (page 690)

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 689).

Availability
Available in iOS 2.0 and later.

See Also
+ systemLocale (page 696)
+ autoupdatingCurrentLocale (page 689)

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 iOS 2.0 and later.

See Also
+ availableLocaleIdentifiers (page 690)
+ ISOLanguageCodes (page 694)
+ ISOCurrencyCodes (page 693)
+ commonISOCurrencyCodes (page 691)

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 iOS 2.0 and later.

See Also
+ availableLocaleIdentifiers (page 690)
+ ISOCountryCodes (page 693)

+ ISOLanguageCodes (page 694)
+ commonISOCurrencyCodes (page 691)

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 iOS 2.0 and later.

See Also
+ availableLocaleIdentifiers (page 690)
+ ISOCountryCodes (page 693)
+ ISOCurrencyCodes (page 693)
+ commonISOCurrencyCodes (page 691)

Declared In
NSLocale.h

lineDirectionForLanguage:

Returns the line direction for the specified ISO language code.

```
+ (NSLocaleLanguageDirection)lineDirectionForLanguage:(NSString *)isoLangCode
```

Parameters
isoLangCode
The ISO language code.

Return Value
Returns the line direction for the language. See "NSLocaleLanguageDirection" (page 699) for possible values. If the appropriate direction can't be determined NSLocaleLanguageDirectionUnknown (page 699) is returned.

Availability
Available in iOS 4.0 and later.

See Also
+ characterDirectionForLanguage: (page 691)

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 699).

Return Value

A locale identifier created from the components specified in *dict*.

Discussion

This reverses the actions of `componentsFromLocaleIdentifier:` (page 692), so for example the dictionary `{NSLocaleLanguageCode="en", NSLocaleCountryCode="US", NSLocaleCalendar=NSJapaneseCalendar}` becomes `"en_US@calendar=japanese"`.

Availability

Available in iOS 2.0 and later.

See Also

`+ componentsFromLocaleIdentifier:` (page 692)

`+ canonicalLocaleIdentifierFromString:` (page 690)

`+ ISOLanguageCodes` (page 694)

Declared In

`NSLocale.h`

localeIdentifierFromWindowsLocaleCode:

Returns a locale identifier from a Windows locale code.

```
+ (NSString *)localeIdentifierFromWindowsLocaleCode:(uint32_t)lcid
```

Parameters

lcid

The Windows locale code.

Return Value

The locale identifier.

Availability

Available in iOS 4.0 and later.

See Also

`+ windowsLocaleCodeFromLocaleIdentifier:` (page 696)

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 iOS 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 iOS 2.0 and later.

See Also
`+ autoupdatingCurrentLocale` (page 689)
`+ autoupdatingCurrentLocale` (page 689)

Declared In
`NSLocale.h`

windowsLocaleCodeFromLocaleIdentifier:

Returns a Window locale code from the locale identifier.

```
+ (uint32_t)windowsLocaleCodeFromLocaleIdentifier:(NSString *)localeIdentifier
```

Parameters
localeIdentifier
The locale identifier.

Return Value
The Windows locale code.

Availability
Available in iOS 4.0 and later.

See Also
`+ localeIdentifierFromWindowsLocaleCode:` (page 695)

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 699)),

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 iOS 2.0 and later.

See Also
- `localeIdentifier` (page 698)

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 iOS 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 iOS 2.0 and later.

See Also
- `displayNameForKey:value:` (page 697)

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 699).

Return Value

The object corresponding to *key*.

Availability

Available in iOS 2.0 and later.

See Also

- displayNameForKey:value: (page 697)

Declared In

NSLocale.h

Constants

NSLocaleLanguageDirection

These constants describe the text direction for a language. Used by the methods lineDirectionForLanguage: (page 694) and characterDirectionForLanguage: (page 691).

```
enum {
    NSLocaleLanguageDirectionUnknown = kCFLocaleLanguageDirectionUnknown,
    NSLocaleLanguageDirectionLeftToRight = kCFLocaleLanguageDirectionLeftToRight,
    NSLocaleLanguageDirectionRightToLeft = kCFLocaleLanguageDirectionRightToLeft,
    NSLocaleLanguageDirectionTopToBottom = kCFLocaleLanguageDirectionTopToBottom,
    NSLocaleLanguageDirectionBottomToTop = kCFLocaleLanguageDirectionBottomToTop
};
typedef NSUInteger NSLocaleLanguageDirection;
```

Constants

NSLocaleLanguageDirectionUnknown

The direction of the language is unknown.

Available in iOS 4.0 and later.

Declared in NSLocale.h.

NSLocaleLanguageDirectionLeftToRight

The language direction is from left to right.

Available in iOS 4.0 and later.

Declared in NSLocale.h.

`NSLocaleLanguageDirectionRightToLeft`
The language direction is from right to left.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

`NSLocaleLanguageDirectionTopToBottom`
The language direction is from top to bottom.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

`NSLocaleLanguageDirectionBottomToTop`
The language direction is from bottom to top.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

NSLocale Component Keys

The following constants specify keys used to retrieve components of a locale with `objectForKey:` (page 699).

```
NSString * const NSLocaleIdentifier;
NSString * const NSLocaleLanguageCode;
NSString * const NSLocaleCountryCode;
NSString * const NSLocaleScriptCode;
NSString * const NSLocaleVariantCode;
NSString * const NSLocaleExemplarCharacterSet;
NSString * const NSLocaleCalendar;
NSString * const NSLocaleCollationIdentifier;
NSString * const NSLocaleUsesMetricSystem;
NSString * const NSLocaleMeasurementSystem;
NSString * const NSLocaleDecimalSeparator;
NSString * const NSLocaleGroupingSeparator;
NSString * const NSLocaleCurrencySymbol;
NSString * const NSLocaleCurrencyCode;
NSString * const NSLocaleCollatorIdentifier;
NSString * const NSLocaleQuotationBeginDelimiterKey;
NSString * const NSLocaleQuotationEndDelimiterKey;
NSString * const NSLocaleAlternateQuotationBeginDelimiterKey;
NSString * const NSLocaleAlternateQuotationEndDelimiterKey;
```

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in `NSLocale.h`.

`NSLocaleCollatorIdentifier`
The key for the collation identifier for the locale.
The corresponding value is an `NSString` object. If unknown, `nil` is returned.
Available in iOS 4.0 and later.
Declared in `NSLocale.h`.

`NSLocaleQuotationBeginDelimiterKey`
The key for the begin quotation symbol associated with the locale.
The corresponding value is an `NSString` object.
Available in iOS 4.0 and later.
Declared in `NSLocale.h`.

`NSLocaleQuotationEndDelimiterKey`
The key for the begin quotation symbol associated with the locale.
The corresponding value is an `NSString` object.
Available in iOS 4.0 and later.
Declared in `NSLocale.h`.

`NSLocaleAlternateQuotationBeginDelimiterKey`
The key for the alternating begin quotation symbol associated with the locale. In some locales, when quotations are nested, the quotation characters alternate. Thus, `NSLocaleQuotationBeginDelimiterKey`, then `NSLocaleAlternateQuotationBeginDelimiterKey`, etc.
The corresponding value is an `NSString` object.
Available in iOS 4.0 and later.
Declared in `NSLocale.h`.

`NSLocaleAlternateQuotationEndDelimiterKey`
The key for the alternating enda quotation symbol associated with the locale. In some locales, when quotations are nested, the quotation characters alternate. Thus, `NSLocaleQuotationEndDelimiterKey`, then `NSLocaleAlternateQuotationEndDelimiterKey`, etc.

The corresponding value is an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

NSLocale Calendar Keys

These constants identify `NSCalendar` instances.

```
NSString * const NSGregorianCalendar;
NSString * const NSBuddhistCalendar;
NSString * const NSChineseCalendar;
NSString * const NSHebrewCalendar;
NSString * const NSIslamicCalendar;
NSString * const NSIslamicCivilCalendar;
NSString * const NSJapaneseCalendar;
NSString * const NSRepublicOfChinaCalendar;
NSString * const NSPersianCalendar;
NSString * const NSIndianCalendar;
NSString * const NSISO8601Calendar;
```

Constants

`NSGregorianCalendar`
Identifier for the Gregorian calendar.

Available in iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSBuddhistCalendar`
Identifier for the Buddhist calendar.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSHebrewCalendar`
Identifier for the Hebrew calendar.

Available in iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSIslamicCalendar`
Identifier for the Islamic calendar.

Available in iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSIslamicCivilCalendar`
Identifier for the Islamic civil calendar.

Available in iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSJapaneseCalendar`
Identifier for the Japanese calendar.

Available in iOS 2.0 and later.

Declared in `NSLocale.h`.

`NSRepublicOfChinaCalendar`
Identifier for the Republic of China (Taiwan) calendar.

A Chinese calendar can be created, and one can do calendrical calculations with it, but it should not be used for formatting as the necessary underlying functionality is not functioning correctly yet.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

`NSPersianCalendar`
Identifier for the Persian calendar

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

`NSIndianCalendar`
Identifier for the Indian calendar

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

`NSISO8601Calendar`
Identifier for the ISO8601. The ISO8601 calendar is not yet implemented.

Available in iOS 4.0 and later.

Declared in `NSLocale.h`.

Discussion
You use these identifiers to initialize a new `NSCalendar` object, using `initWithCalendarIdentifier:` (page 173). You get one of these identifiers as the return value from `calendarIdentifier` (page 168).

Declared In
`NSLocale.h`

Notifications

NSCurrentLocaleDidChangeNotification

Notification that indicates that the user's locale changed.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

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 1609)
- `unlock` (page 1610)

Tasks

Acquiring a Lock

- `lockBeforeDate:` (page 708)
  Attempts to acquire a lock before a given time and returns a Boolean value indicating whether the attempt was successful.
- `tryLock` (page 709)
  Attempts to acquire a lock and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

- `setName:` (page 709)
  Assigns a name to the receiver.
- `name` (page 708)
  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 iOS 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 iOS 2.0 and later.

See Also
- `setName:` (page 709)

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 iOS 2.0 and later.

See Also
- `name` (page 708)

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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

`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 712)
Creates and returns a port object configured with the given Mach port.

`+ portWithMachPort:options:` (page 713)
Creates and returns a port object configured with the specified options and the given Mach port.

- `initWithMachPort:` (page 714)
  Initializes a newly allocated `NSMachPort` object with a given Mach port.
- `initWithMachPort:options:` (page 714)
  Initializes a newly allocated `NSMachPort` object with a given Mach port and the specified options.

Getting the Mach Port

- `machPort` (page 715)
  Returns as an `int` the Mach port used by the receiver.

Scheduling the Port on a Run Loop

- `removeFromRunLoop:forMode:` (page 715)
  Removes the receiver from the run loop mode *`mode`* of *`runLoop`*.
- `scheduleInRunLoop:forMode:` (page 715)
  Schedules the receiver into the run loop mode *`mode`* of *`runLoop`*.

Getting and Setting the Delegate

- `delegate` (page 713)
  Returns the receiver's delegate.
- `setDelegate:` (page 716)
  Sets the receiver's delegate to a given object.

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 iOS 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 716).

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 iOS 2.0 and later.

Declared In
NSPort.h

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id < NSMachPortDelegate >)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iOS 4.0 and later.

See Also
- setDelegate: (page 716)

Declared In
NSPort.h

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 iOS 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 716).

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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- scheduleInRunLoop:forMode:` (page 715)

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 1611).

Availability
Available in iOS 2.0 and later.

See Also
`- removeFromRunLoop:forMode:` (page 715)

Declared In
`NSPort.h`

setDelegate:

Sets the receiver's delegate to a given object.

```
- (void)setDelegate:(id < NSMachPortDelegate >)anObject
```

Parameters
anObject
The delegate for the receiver.

Availability
Available in iOS 4.0 and later.

See Also
`- delegate` (page 713)

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 iOS 2.0 and later.
Declared in `NSPort.h`.

`NSMachPortDeallocateSendRight`

Deallocate a send right when the `NSMachPort` object is invalidated or destroyed.

Available in iOS 2.0 and later.

Declared in `NSPort.h`.

`NSMachPortDeallocateReceiveRight`

Remove a receive right when the `NSMachPort` object is invalidated or destroyed.

Available in iOS 2.0 and later.

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 iOS 2.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

`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.

Important: Avoid `NSMessagePort`. There's little reason to use `NSMessagePort` rather than `NSMachPort` or `NSSocketPort`. There's no particular performance or functionality advantage. It is recommended avoiding its use.

`NSMessagePort` may be deprecated in the Mac OS X v 10.6 or later.

NSMethodSignature Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSMethodSignature.h
Companion guides	Distributed Objects Programming Topics The Objective-C Programming Language

Overview

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. You typically create an `NSMethodSignature` object using `NSObject`'s `methodSignatureForSelector:` (page 974) instance method (on Mac OS X v10.5 and later you can also use `signatureWithObjCTypes:` (page 722)). It is then used to create an `NSInvocation` object, which is passed as the argument to a `forwardInvocation:` (page 970) message to send the invocation on to whatever other object can handle the message. In the default case, `NSObject` invokes `doesNotRecognizeSelector:` (page 967), 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 723) 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 725), the total stack frame length occupied by all arguments with `frameLength` (page 723) (this varies with hardware architecture), and the length and type of the return value with `methodReturnLength` (page 724) and `methodReturnType` (page 725). Finally, applications using distributed objects can determine if the method is asynchronous with the `isOneway` (page 724) method.

For more information about the nature of a method, including the hidden arguments, see "How Messaging Works" in *The Objective-C Programming Language*.

Tasks

Creating a Method Signature Object

`+ signatureWithObjCTypes:` (page 722)
Returns an `NSMethodSignature` object for the given Objective C method type string.

Getting Information on Argument Types

`- getArgumentTypeAtIndex:` (page 723)
Returns the type encoding for the argument at a given index.

`- numberOfArguments` (page 725)
Returns the number of arguments recorded in the receiver.

`- frameLength` (page 723)
Returns the number of bytes that the arguments, taken together, occupy on the stack.

Getting Information on Return Types

`- methodReturnType` (page 725)
Returns a C string encoding the return type of the method in Objective-C type encoding.

`- methodReturnLength` (page 724)
Returns the number of bytes required for the return value.

Determining Synchronous Status

`- isOneway` (page 724)
Returns a Boolean value that indicates whether the receiver is asynchronous when invoked through distributed objects.

Class Methods

signatureWithObjCTypes:

Returns an `NSMethodSignature` object for the given Objective C method type string.

```
+ (NSMethodSignature *)signatureWithObjCTypes:(const char *)types
```

Parameters

types
An array of characters containing the type encodings for the method arguments.

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`.

Return Value
An `NSMethodSignature` object for the given Objective C method type string in *types*.

Discussion

Special Considerations

This method, available since Mac OS X v10.0, is exposed in Mac OS X v10.5. Only type encoding strings of the style of the runtime that the application is running against are supported. In exposing this method there is no commitment to binary compatibily supporting any "old-style" type encoding strings after such changes occur.

It is your responsibility to pass in type strings which are either from the current runtime data or match the style of type string in use by the runtime that the application is running on.

Availability
Available in iOS 2.0 and later.

Declared In
`NSMethodSignature.h`

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`- methodReturnType` (page 725)

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 iOS 2.0 and later.

See Also
`- methodReturnLength` (page 724)

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 iOS 2.0 and later.

Declared In
`NSMethodSignature.h`

NSMutableArray Class Reference

Inherits from	NSArray : NSObject
Conforms to	NSCoding (NSArray) NSCopying (NSArray) NSMutableCopying (NSArray) NSFastEnumeration (NSArray) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSArray.h Foundation/NSPredicate.h Foundation/NSSortDescriptor.h
Companion guides	Collections Programming Topics Key-Value Coding Programming Guide
Related sample code	BonjourWeb CryptoExercise GKRocket ScrollViewSuite WiTap

Overview

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 733)
- `removeObjectAtIndex:` (page 737)
- `addObject:` (page 731)
- `removeLastObject` (page 736)
- `replaceObjectAtIndex:withObject:` (page 741)

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 1638) message. When an object is removed from a mutable array, it receives a `release` (page 1636) 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 1638) 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];
```

Filtering using a predicate: The `filterUsingPredicate:` (page 732) method provides in-place in-memory filtering of an array using an `NSPredicate` object. 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. This method and the `NSPredicate` class are not available in iOS prior to v3.0.

Tasks

Creating and Initializing a Mutable Array

`+ arrayWithCapacity:` (page 730)
Creates and returns an `NSMutableArray` object with enough allocated memory to initially hold a given number of objects.

`- initWithCapacity:` (page 732)
Returns an array, initialized with enough memory to initially hold a given number of objects.

Adding Objects

`- addObject:` (page 731)
Inserts a given object at the end of the receiver.

`- addObjectsFromArray:` (page 731)
Adds the objects contained in another given array to the end of the receiver's content.

`- insertObject:atIndex:` (page 733)
Inserts a given object into the receiver's contents at a given index.

`- insertObjects:atIndexes:` (page 734)
Inserts the objects in in a given array into the receiver at the specified indexes.

Removing Objects

- `removeAllObjects` (page 735)
  Empties the receiver of all its elements.
- `removeLastObject` (page 736)
  Removes the object with the highest-valued index in the receiver
- `removeObject:` (page 736)
  Removes all occurrences in the receiver of a given object.
- `removeObject:inRange:` (page 737)
  Removes all occurrences within a specified range in the receiver of a given object.
- `removeObjectAtIndex:` (page 737)
  Removes the object at *index*.
- `removeObjectsAtIndexes:` (page 739)
  Removes the objects at the specified indexes from the receiver.
- `removeObjectIdenticalTo:` (page 738)
  Removes all occurrences of a given object in the receiver.
- `removeObjectIdenticalTo:inRange:` (page 739)
  Removes all occurrences of *anObject* within the specified range in the receiver.
- `removeObjectsInArray:` (page 741)
  Removes from the receiver the objects in another given array.
- `removeObjectsInRange:` (page 741)
  Removes from the receiver each of the objects within a given range.
- `removeObjectsFromIndices:numIndices:` (page 740) Deprecated in iOS 4.0
  Removes the specified number of objects from the receiver, beginning at the specified index. (Deprecated. Use `removeObjectsAtIndexes:` (page 739) instead.)

Replacing Objects

- `replaceObjectAtIndex:withObject:` (page 741)
  Replaces the object at *index* with *anObject*.
- `replaceObjectsAtIndexes:withObjects:` (page 742)
  Replaces the objects in the receiver at specified locations specified with the objects from a given array.
- `replaceObjectsInRange:withObjectsFromArray:range:` (page 743)
  Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.
- `replaceObjectsInRange:withObjectsFromArray:` (page 743)
  Replaces the objects in the receiver specified by a given range with all of the objects from a given array.
- `setArray:` (page 744)
  Sets the receiver's elements to those in another given array.

Filtering Content

- `filterUsingPredicate:` (page 732)
  Evaluates a given predicate against the receiver's content and leaves only objects that match

Rearranging Content

- `exchangeObjectAtIndex:withObjectAtIndex:` (page 732)
  Exchanges the objects in the receiver at given indices.
- `sortUsingDescriptors:` (page 745)
  Sorts the receiver using a given array of sort descriptors.
- `sortUsingComparator:` (page 744)
  Sorts the receiver using the comparison method specified by a given `NSComparator` Block.
- `sortWithOptions:usingComparator:` (page 746)
  Sorts the receiver using the specified options and the comparison method specified by a given `NSComparator` Block.
- `sortUsingFunction:context:` (page 745)
  Sorts the receiver's elements in ascending order as defined by the comparison function *`compare`*.
- `sortUsingSelector:` (page 746)
  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 iOS 2.0 and later.

See Also
- `initWithCapacity:` (page 732)

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 iOS 2.0 and later.

See Also
- `addObjectsFromArray:` (page 731)
- `removeObject:` (page 736)
- `setArray:` (page 744)

Related Sample Code
ScrollViewSuite
SpeakHere

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 iOS 2.0 and later.

See Also
- `setArray:` (page 744)
- `removeObject:` (page 736)

Related Sample Code
AddMusic

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 iOS 2.0 and later.

Declared In
NSArray.h

filterUsingPredicate:

Evaluates a given predicate against the receiver's content and leaves only objects that match

```
- (void)filterUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate

The predicate to evaluate against the receiver's elements.

Availability
Available in iOS 3.0 and later.

See Also
- filteredArrayUsingPredicate: (page 56) (NSArray)

Declared In
NSPredicate.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 iOS 2.0 and later.

See Also
+ arrayWithCapacity: (page 730)

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 iOS 2.0 and later.

See Also
- removeObjectAtIndex: (page 737)

Related Sample Code
MultipleDetailViews

ToolbarSearch

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
{
    NSUInteger currentIndex = [indexes firstIndex];
    NSUInteger 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)`.

If *`objects`* or `indexes` is `nil` this method will raise an exception.

Availability
Available in iOS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 733)

Declared In
`NSArray.h`

removeAllObjects

Empties the receiver of all its elements.

```
- (void)removeAllObjects
```

Availability
Available in iOS 2.0 and later.

See Also
- `removeObject:` (page 736)
- `removeLastObject` (page 736)
- `removeObjectAtIndex:` (page 737)
- `removeObjectIdenticalTo:` (page 738)

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 iOS 2.0 and later.

See Also
- `removeAllObjects` (page 735)
- `removeObject:` (page 736)
- `removeObjectAtIndex:` (page 737)
- `removeObjectIdenticalTo:` (page 738)

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 61) to locate matches and then removes them by using `removeObjectAtIndex:` (page 737). 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 iOS 2.0 and later.

See Also
- `removeAllObjects` (page 735)
- `removeLastObject` (page 736)
- `removeObjectAtIndex:` (page 737)
- `removeObjectIdenticalTo:` (page 738)
- `removeObjectsInArray:` (page 741)

Related Sample Code
MultipleDetailViews

ToolbarSearch

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 iOS 2.0 and later.

See Also
- `removeAllObjects` (page 735)
- `removeLastObject` (page 736)
- `removeObjectAtIndex:` (page 737)
- `removeObjectIdenticalTo:` (page 738)
- `removeObjectsInArray:` (page 741)

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 iOS 2.0 and later.

See Also

- `insertObject:atIndex:` (page 733)
- `removeAllObjects` (page 735)
- `removeLastObject` (page 736)
- `removeObject:` (page 736)
- `removeObjectIdenticalTo:` (page 738)
- `removeObjectsAtIndexes:` (page 739)

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 65) method to locate matches and then removes them by using `removeObjectAtIndex:` (page 737). 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 iOS 2.0 and later.

See Also

- `removeAllObjects` (page 735)
- `removeLastObject` (page 736)
- `removeObject:` (page 736)
- `removeObjectAtIndex:` (page 737)

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 65) method to locate matches and then removes them by using `removeObjectAtIndex:` (page 737). 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 iOS 2.0 and later.

See Also

- `removeAllObjects` (page 735)
- `removeLastObject` (page 736)
- `removeObject:` (page 736)
- `removeObjectAtIndex:` (page 737)
- `removeObjectsAtIndexes:` (page 739)

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 737), 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)
```

If `indexes` is `nil` this method will raise an exception.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithCapacity:` (page 732)
- `removeObjectAtIndex:` (page 737)
- `removeObject:inRange:` (page 737)

Declared In
`NSArray.h`

removeObjectsFromIndices:numIndices:

Removes the specified number of objects from the receiver, beginning at the specified index. (Deprecated in iOS 4.0. Use `removeObjectsAtIndexes:` (page 739) instead.)

```
- (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 737), 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 iOS 2.0 and later.
Deprecated in iOS 4.0.

See Also
- `initWithCapacity:` (page 732)
- `removeObjectAtIndex:` (page 737)
- `removeObject:inRange:` (page 737)
- `removeObjectsAtIndexes:` (page 739)

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 736), 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 iOS 2.0 and later.

See Also

- `removeAllObjects` (page 735)
- `removeObjectIdenticalTo:` (page 738)
- `removeObjectsAtIndexes:` (page 739)

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 737).

Availability

Available in iOS 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 iOS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 733)
- `removeObjectAtIndex:` (page 737)
- `removeObjectsAtIndexes:` (page 739)
- `replaceObjectsAtIndexes:withObjects:` (page 742)

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*.

If *objects* or `indexes` is `nil` this method will raise an exception.

Availability
Available in iOS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 733)
- `removeObjectAtIndex:` (page 737)
- `replaceObjectAtIndex:withObject:` (page 741)

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 iOS 2.0 and later.

See Also
- insertObject:atIndex: (page 733)
- removeObjectAtIndex: (page 737)
- replaceObjectAtIndex:withObject: (page 741)
- replaceObjectsAtIndexes:withObjects: (page 742)

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 iOS 2.0 and later.

See Also
- `insertObject:atIndex:` (page 733)
- `removeObjectAtIndex:` (page 737)
- `replaceObjectAtIndex:withObject:` (page 741)
- `replaceObjectsAtIndexes:withObjects:` (page 742)

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 iOS 2.0 and later.

See Also
- `addObjectsFromArray:` (page 731)
- `insertObject:atIndex:` (page 733)

Declared In
`NSArray.h`

sortUsingComparator:

Sorts the receiver using the comparison method specified by a given `NSComparator` Block.

```
- (void)sortUsingComparator:(NSComparator)cmptr
```

Parameters
cmptr
A comparator block.

Availability
Available in iOS 4.0 and later.

See Also
- `sortUsingFunction:context:` (page 745)

- sortUsingSelector: (page 746)
- sortUsingDescriptors: (page 745)
- sortWithOptions:usingComparator: (page 746)
- sortedArrayUsingDescriptors: (page 77) (NSArray)

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 iOS 2.0 and later.

See Also
- sortUsingFunction:context: (page 745)
- sortUsingSelector: (page 746)
- sortUsingComparator: (page 744)
- sortWithOptions:usingComparator: (page 746)
- sortedArrayUsingDescriptors: (page 77) (NSArray)

Declared In
NSSortDescriptor.h

sortUsingFunction:context:

Sorts the receiver's elements in ascending order as defined by the comparison function *compare*.

```
- (void)sortUsingFunction:(NSInteger (*)(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 iOS 2.0 and later.

See Also
- `sortUsingDescriptors:` (page 745)
- `sortUsingSelector:` (page 746)
- `sortedArrayUsingFunction:context:` (page 78) (`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 iOS 2.0 and later.

See Also
- `sortUsingDescriptors:` (page 745)
- `sortUsingFunction:context:` (page 745)
- `sortedArrayUsingSelector:` (page 80) (`NSArray`)

Declared In
`NSArray.h`

sortWithOptions:usingComparator:

Sorts the receiver using the specified options and the comparison method specified by a given `NSComparator` Block.

```
- (void)sortWithOptions:(NSSortOptions)opts usingComparator:(NSComparator)cmptr
```

Parameters

opts

A bitmask that specifies the options for the sort (whether it should be performed concurrently and whether it should be performed stably).

cmptr

A comparator block.

Availability

Available in iOS 4.0 and later.

See Also

- `sortUsingFunction:context:` (page 745)
- `sortUsingSelector:` (page 746)
- `sortUsingDescriptors:` (page 745)
- `sortUsingComparator:` (page 744)
- `sortedArrayUsingDescriptors:` (page 77) (`NSArray`)

Declared In

`NSArray.h`

NSMutableAttributedString Class Reference

Inherits from	NSAttributedString : NSObject
Conforms to	NSCoding (NSAttributedString) NSCopying (NSAttributedString) NSMutableCopying (NSAttributedString) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.2 and later.
Declared in	Foundation/NSAttributedString.h
Companion guide	Attributed String Programming Guide

Overview

`NSMutableAttributedString` declares the programmatic interface to objects that manage mutable attributed strings. You can add and remove characters (raw strings) and attributes separately or together as attributed strings. See the class description for `NSAttributedString` for more information about attributed strings.

When working with the Application Kit, you must also clean up changed attributes using the various `fix...` methods. See "Changing an Attributed String" for more information on fixing attributes. These methods, as well as others involving setting graphical attributes, are described in NSMutableAttributedString Additions in the Application Kit.

`NSMutableAttributedString` adds two primitive methods to those of `NSAttributedString`. These primitive methods provide the basis for all the other methods in its class. The primitive `replaceCharactersInRange:withString:` (page 755) method replaces a range of characters with those from a string, leaving all attribute information outside that range intact. The primitive `setAttributes:range:` (page 756) method sets attributes and values for a given range of characters, replacing any previous attributes and values for that range.

In Mac OS X, the Application Kit also uses NSParagraphStyle and its subclass NSMutableParagraphStyle to encapsulate the paragraph or ruler attributes used by the `NSAttributedString` classes.

Note that the default font for `NSAttributedString` objects is Helvetica 12-point, which differs from the Mac OS X system font Lucida Grande, so you may wish to create the string with non-default attributes suitable for your application using, for example, `initWithString:attributes:` (page 98).

iOS Note: In iOS, this class is used primarily in conjunction with the Core Text framework.

Tasks

Retrieving Character Information

- `mutableString` (page 754)
  Returns the character contents of the receiver as an `NSMutableString` object.

Changing Characters

- `replaceCharactersInRange:withString:` (page 755)
  Replaces the characters in the given range with the characters of the given string.
- `deleteCharactersInRange:` (page 753)
  Deletes the characters in the given range along with their associated attributes.

Changing Attributes

- `setAttributes:range:` (page 756)
  Sets the attributes for the characters in the specified range to the specified attributes.
- `addAttribute:value:range:` (page 751)
  Adds an attribute with the given name and value to the characters in the specified range.
- `addAttributes:range:` (page 751)
  Adds the given collection of attributes to the characters in the specified range.
- `removeAttribute:range:` (page 754)
  Removes the named attribute from the characters in the specified range.

Changing Characters and Attributes

- `appendAttributedString:` (page 752)
  Adds the characters and attributes of a given attributed string to the end of the receiver.
- `insertAttributedString:atIndex:` (page 753)
  Inserts the characters and attributes of the given attributed string into the receiver at the given index.
- `replaceCharactersInRange:withAttributedString:` (page 755)
  Replaces the characters and attributes in a given range with the characters and attributes of the given attributed string.
- `setAttributedString:` (page 756)
  Replaces the receiver's entire contents with the characters and attributes of the given attributed string.

Grouping Changes

- `beginEditing` (page 752)
  Overridden by subclasses to buffer or optimize a series of changes to the receiver's characters or attributes, until it receives a matching `endEditing` (page 753) message, upon which it can consolidate changes and notify any observers that it has changed.
- `endEditing` (page 753)
  Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 752) message and to notify any observers of the changes.

Instance Methods

addAttribute:value:range:

Adds an attribute with the given name and value to the characters in the specified range.

```
- (void)addAttribute:(NSString *)name value:(id)value range:(NSRange)aRange
```

Parameters

name
  A string specifying the attribute name. Attribute keys can be supplied by another framework or can be custom ones you define. For information about where to find the system-supplied attribute keys, see the overview section in *NSAttributedString Class Reference*.

value
  The attribute value associated with *name*.

aRange
  The range of characters to which the specified attribute/value pair applies.

Discussion
You may assign any *name*/*value* pair you wish to a range of characters, in addition to the standard attributes described in the "Constants" section of NSAttributedString Additions. Raises an `NSInvalidArgumentException` if *name* or *value* is `nil` and an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability
Available in iOS 3.2 and later.

See Also
- `addAttributes:range:` (page 751)
- `removeAttribute:range:` (page 754)

Declared In
`NSAttributedString.h`

addAttributes:range:

Adds the given collection of attributes to the characters in the specified range.

```
- (void)addAttributes:(NSDictionary *)attributes range:(NSRange)aRange
```

Parameters

attributes

A dictionary containing the attributes to add. Attribute keys can be supplied by another framework or can be custom ones you define. For information about where to find the system-supplied attribute keys, see the overview section in *NSAttributedString Class Reference*.

aRange

The range of characters to which the specified attributes apply.

Discussion

You may assign any name/value pair you wish to a range of characters, in addition to the standard attributes described in the "Constants" section of NSAttributedString Additions. Raises an `NSInvalidArgumentException` if *attributes* is `nil` and an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability

Available in iOS 3.2 and later.

See Also

- `addAttribute:value:range:` (page 751)
- `removeAttribute:range:` (page 754)

Declared In

`NSAttributedString.h`

appendAttributedString:

Adds the characters and attributes of a given attributed string to the end of the receiver.

```
- (void)appendAttributedString:(NSAttributedString *)attributedString
```

Parameters

attributedString

The string whose characters and attributes are added.

Availability

Available in iOS 3.2 and later.

See Also

- `insertAttributedString:atIndex:` (page 753)

Declared In

`NSAttributedString.h`

beginEditing

Overridden by subclasses to buffer or optimize a series of changes to the receiver's characters or attributes, until it receives a matching `endEditing` (page 753) message, upon which it can consolidate changes and notify any observers that it has changed.

```
- (void)beginEditing
```

Discussion

You can nest pairs of `beginEditing` and `endEditing` (page 753) messages.

Availability
Available in iOS 3.2 and later.

Declared In
NSAttributedString.h

deleteCharactersInRange:

Deletes the characters in the given range along with their associated attributes.

```
- (void)deleteCharactersInRange:(NSRange)aRange
```

Parameters

aRange

A range specifying the characters to delete.

Discussion
Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability
Available in iOS 3.2 and later.

See Also
- `replaceCharactersInRange:withAttributedString:` (page 755)
- `replaceCharactersInRange:withString:` (page 755)

Declared In
NSAttributedString.h

endEditing

Overridden by subclasses to consolidate changes made since a previous `beginEditing` (page 752) message and to notify any observers of the changes.

```
- (void)endEditing
```

Discussion
The `NSMutableAttributedString` implementation does nothing. `NSTextStorage`, for example, overrides this method to invoke `fixAttributesInRange:` and to inform its `NSLayoutManager` objects that they need to re-lay the text.

Availability
Available in iOS 3.2 and later.

Declared In
NSAttributedString.h

insertAttributedString:atIndex:

Inserts the characters and attributes of the given attributed string into the receiver at the given index.

```
- (void)insertAttributedString:(NSAttributedString *)attributedString
    atIndex:(NSUInteger)index
```

Parameters

attributedString

The string whose characters and attributes are inserted.

index

The index at which the characters and attributes are inserted.

Discussion

The new characters and attributes begin at the given index and the existing characters and attributes from the index to the end of the receiver are shifted by the length of the attributed string. Raises an `NSRangeException` if *index* lies beyond the end of the receiver's characters.

Availability

Available in iOS 3.2 and later.

See Also

- appendAttributedString: (page 752)

Declared In

NSAttributedString.h

mutableString

Returns the character contents of the receiver as an `NSMutableString` object.

```
- (NSMutableString *)mutableString
```

Return Value

The mutable string object.

Discussion

The receiver tracks changes to this string and keeps its attribute mappings up to date.

Availability

Available in iOS 3.2 and later.

Declared In

NSAttributedString.h

removeAttribute:range:

Removes the named attribute from the characters in the specified range.

```
- (void)removeAttribute:(NSString *)name range:(NSRange)aRange
```

Parameters

name

A string specifying the attribute name to remove. Attribute keys can be supplied by another framework or can be custom ones you define. For information about where to find the system-supplied attribute keys, see the overview section in *NSAttributedString Class Reference*.

aRange
The range of characters from which the specified attribute is removed.

Discussion
Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability
Available in iOS 3.2 and later.

See Also
- `addAttribute:value:range:` (page 751)
- `addAttributes:range:` (page 751)

Declared In
`NSAttributedString.h`

replaceCharactersInRange:withAttributedString:

Replaces the characters and attributes in a given range with the characters and attributes of the given attributed string.

```
- (void)replaceCharactersInRange:(NSRange)aRange
    withAttributedString:(NSAttributedString *)attributedString
```

Parameters
aRange
The range of characters and attributes replaced.

attributedString
The attributed string whose characters and attributes replace those in the specified range.

Discussion
Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability
Available in iOS 3.2 and later.

See Also
- `insertAttributedString:atIndex:` (page 753)

Declared In
`NSAttributedString.h`

replaceCharactersInRange:withString:

Replaces the characters in the given range with the characters of the given string.

```
- (void)replaceCharactersInRange:(NSRange)aRange withString:(NSString *)aString
```

Parameters
aRange
A range specifying the characters to replace.

aString
A string specifying the characters to replace those in `aRange`.

Discussion
The new characters inherit the attributes of the first replaced character from *aRange*. Where the length of *aRange* is 0, the new characters inherit the attributes of the character preceding *aRange* if it has any, otherwise of the character following *aRange*.

Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

Availability
Available in iOS 3.2 and later.

See Also
`- deleteCharactersInRange:` (page 753)

Declared In
`NSAttributedString.h`

setAttributedString:

Replaces the receiver's entire contents with the characters and attributes of the given attributed string.

```
- (void)setAttributedString:(NSAttributedString *)attributedString
```

Parameters
attributedString
The attributed string whose characters and attributes replace those in the receiver.

Availability
Available in iOS 3.2 and later.

See Also
`- appendAttributedString:` (page 752)

Declared In
`NSAttributedString.h`

setAttributes:range:

Sets the attributes for the characters in the specified range to the specified attributes.

```
- (void)setAttributes:(NSDictionary *)attributes range:(NSRange)aRange
```

Parameters
attributes
A dictionary containing the attributes to set. Attribute keys can be supplied by another framework or can be custom ones you define. For information about where to find the system-supplied attribute keys, see the overview section in *NSAttributedString Class Reference*.

aRange
The range of characters whose attributes are set.

Discussion
These new attributes replace any attributes previously associated with the characters in *aRange*. Raises an `NSRangeException` if any part of *aRange* lies beyond the end of the receiver's characters.

To set attributes for a zero-length `NSMutableAttributedString` displayed in a text view, use the `NSTextView` method `setTypingAttributes:`.

Availability
Available in iOS 3.2 and later.

See Also
- `addAttributes:range:` (page 751)
- `removeAttribute:range:` (page 754)

Declared In
`NSAttributedString.h`

NSMutableCharacterSet Class Reference

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

Overview

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 1614).

Tasks

Adding and Removing Characters

- `addCharactersInRange:` (page 760)
  Adds to the receiver the characters whose Unicode values are in a given range.
- `removeCharactersInRange:` (page 762)
  Removes from the receiver the characters whose Unicode values are in a given range.

- `addCharactersInString:` (page 761)
  Adds to the receiver the characters in a given string.
- `removeCharactersInString:` (page 763)
  Removes from the receiver the characters in a given string.

Combining Character Sets

- `formIntersectionWithCharacterSet:` (page 761)
  Modifies the receiver so it contains only characters that exist in both the receiver and *otherSet*.
- `formUnionWithCharacterSet:` (page 761)
  Modifies the receiver so it contains all characters that exist in either the receiver or *otherSet*.

Inverting a Character Set

- `invert` (page 762)
  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 iOS 2.0 and later.

See Also

- `removeCharactersInRange:` (page 762)
- `addCharactersInString:` (page 761)

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 iOS 2.0 and later.

See Also
- removeCharactersInString: (page 763)
- addCharactersInRange: (page 760)

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 iOS 2.0 and later.

See Also
- formUnionWithCharacterSet: (page 761)

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 iOS 2.0 and later.

See Also
- `formIntersectionWithCharacterSet:` (page 761)

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 196), is much less efficient than inverting an immutable character set with `invertedSet`.

Availability
Available in iOS 2.0 and later.

See Also
- `invertedSet` (page 196) (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 iOS 2.0 and later.

See Also
- `addCharactersInRange:` (page 760)
- `removeCharactersInString:` (page 763)

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 iOS 2.0 and later.

See Also

- addCharactersInString: (page 761)
- removeCharactersInRange: (page 762)

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 iOS 2.0 and later.
Declared in	Foundation/NSData.h Foundation/NSSerialization.h (Deprecated)
Companion guide	Binary Data Programming Guide
Related sample code	CryptoExercise GKRocket

Overview

`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 767)
Creates and returns an `NSMutableData` object capable of holding the specified number of bytes.

`+ dataWithLength:` (page 767)
Creates and returns an `NSMutableData` object containing a given number of zeroed bytes.

`- initWithCapacity:` (page 769)
Returns an initialized `NSMutableData` object capable of holding the specified number of bytes.

`- initWithLength:` (page 770)
Initializes and returns an `NSMutableData` object containing a given number of zeroed bytes.

Adjusting Capacity

`- increaseLengthBy:` (page 769)
Increases the length of the receiver by a given number of bytes.

`- setLength:` (page 773)
Extends or truncates a mutable data object to a given length.

Accessing Data

`- mutableBytes` (page 770)
Returns a pointer to the receiver's data.

Adding Data

`- appendBytes:length:` (page 768)
Appends to the receiver a given number of bytes from a given buffer.

`- appendData:` (page 768)
Appends the content of another `NSData` object to the receiver.

Modifying Data

`- replaceBytesInRange:withBytes:` (page 771)
Replaces with a given set of bytes a given range within the contents of the receiver.

`- replaceBytesInRange:withBytes:length:` (page 771)
Replaces with a given set of bytes a given range within the contents of the receiver.

`- resetBytesInRange:` (page 772)
Replaces with zeroes the contents of the receiver in a given range.

- setData: (page 772)
  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 iOS 2.0 and later.

See Also
+ dataWithLength: (page 767)
- initWithCapacity: (page 769)
- initWithLength: (page 770)

Related Sample Code
GKRocket

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 iOS 2.0 and later.

See Also
`+ dataWithCapacity:` (page 767)
`- initWithCapacity:` (page 769)
`- initWithLength:` (page 770)

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

See Also
`- appendData:` (page 768)

Related Sample Code
CryptoExercise
GKRocket

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 iOS 2.0 and later.

See Also
- appendBytes:length: (page 768)

Related Sample Code
CryptoExercise
GKRocket

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 iOS 2.0 and later.

See Also
- setLength: (page 773)

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 iOS 2.0 and later.

See Also
+ dataWithCapacity: (page 767)
- initWithLength: (page 770)

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 iOS 2.0 and later.

See Also
+ dataWithCapacity: (page 767)
+ dataWithLength: (page 767)
- initWithCapacity: (page 769)

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 iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

See Also
- `replaceBytesInRange:withBytes:length:` (page 771)
- `resetBytesInRange:` (page 772)

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 iOS 2.0 and later.

See Also

`- replaceBytesInRange:withBytes:` (page 771)

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 iOS 2.0 and later.

See Also

`- replaceBytesInRange:withBytes:` (page 771)

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 771).

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`- increaseLengthBy:` (page 769)

Declared In

`NSData.h`

NSMutableDictionary Class Reference

Inherits from	NSDictionary : NSObject
Conforms to	NSCoding (NSDictionary) NSCopying (NSDictionary) NSMutableCopying (NSDictionary) NSFastEnumeration (NSDictionary) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDictionary.h
Companion guide	Collections Programming Topics
Related sample code	CryptoExercise

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 777)
Returns an empty dictionary with enough allocated space to hold a specified number of objects.

Commonly Used Methods

`removeObjectForKey:` (page 779)
Removes the specified entry from the dictionary.

`removeObjectsForKeys:` (page 780)
Removes multiple entries from the dictionary.

Overview

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 780) and `removeObjectForKey:` (page 779)—this class adds modification operations to the basic operations it inherits from `NSDictionary`.

In a subclass, you must override both of these methods. However, there should be little need of subclassing. If you need to customize behavior, it is often better to consider composition instead of subclassing.

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 1636) 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 777)
Creates and returns a mutable dictionary, initially giving it enough allocated memory to hold a given number of entries.

`- initWithCapacity:` (page 778)
Initializes a newly allocated mutable dictionary, allocating enough memory to hold *numItems* entries.

Adding Entries to a Mutable Dictionary

`- setObject:forKey:` (page 780)
Adds a given key-value pair to the receiver.

`- setValue:forKey:` (page 781)
Adds a given key-value pair to the receiver.

`- addEntriesFromDictionary:` (page 778)
Adds to the receiver the entries from another dictionary.

`- setDictionary:` (page 780)
Sets the contents of the receiver to entries in a given dictionary.

Removing Entries From a Mutable Dictionary

- `removeObjectForKey:` (page 779)
  Removes a given key and its associated value from the receiver.
- `removeAllObjects` (page 779)
  Empties the receiver of its entries.
- `removeObjectsForKeys:` (page 780)
  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 iOS 2.0 and later.

See Also
`dictionary` (page 389) (NSDictionary)
`dictionaryWithContentsOfFile:` (page 390) (NSDictionary)
`dictionaryWithContentsOfURL:` (page 390): (NSDictionary)
`dictionaryWithObject:forKey:` (page 391) (NSDictionary)
`dictionaryWithObjects:forKeys:` (page 392): (NSDictionary)
`dictionaryWithObjects:forKeys:count:` (page 392) (NSDictionary)
`dictionaryWithObjectsAndKeys:` (page 393) (NSDictionary)
`- initWithCapacity:` (page 778)

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 1638) message before being added to the receiver. In contrast, each key object is copied (using `copyWithZone:` (page 1554)—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 iOS 2.0 and later.

See Also

`- setObject:forKey:` (page 780)

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 iOS 2.0 and later.

See Also

`+ dictionaryWithCapacity:` (page 777)

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 1636) message.

Availability
Available in iOS 2.0 and later.

See Also
- `removeObjectForKey:` (page 779)
- `removeObjectsForKeys:` (page 780)

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"];
```

Important: Important: Raises an `NSInvalidArgumentException` (page 1773) if *aKey* is nil.

Availability
Available in iOS 2.0 and later.

See Also
- `removeAllObjects` (page 779)
- `removeObjectsForKeys:` (page 780)

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 iOS 2.0 and later.

See Also
- removeObjectForKey: (page 779)
- removeObjectForKey: (page 779)

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 779)), then each entry from *otherDictionary* added into the receiver.

Availability
Available in iOS 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 1554); 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 1636) message and *anObject* takes its place.

Availability

Available in iOS 2.0 and later.

See Also

`- removeObjectForKey:` (page 779)

Related Sample Code

CryptoExercise

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 780), unless *value* is `nil` in which case the method instead attempts to remove *key* using `removeObjectForKey:` (page 779).

Availability

Available in iOS 2.0 and later.

See Also

`valueForKey:` (page 415) (NSDictionary)

Declared In

`NSKeyValueCoding.h`

NSMutableIndexSet Class Reference

Inherits from	NSIndexSet : NSObject
Conforms to	NSCoding (NSIndexSet) NSCopying (NSIndexSet) NSMutableCopying (NSIndexSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSIndexSet.h
Companion guide	Collections Programming Topics

Overview

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 784)
  Adds an index to the receiver.
- `addIndexes:` (page 784)
  Adds the indexes in an index set to the receiver.
- `addIndexesInRange:` (page 785)
  Adds the indexes in an index range to the receiver.

Removing Indexes

- `removeIndex:` (page 786)
  Removes an index from the receiver.
- `removeIndexes:` (page 786)
  Removes the indexes in an index set from the receiver.
- `removeAllIndexes` (page 785)
  Removes the receiver's indexes.
- `removeIndexesInRange:` (page 786)
  Removes the indexes in an index range from the receiver.

Shifting Index Groups

- `shiftIndexesStartingAtIndex:by:` (page 787)
  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 iOS 2.0 and later.

See Also
- `addIndexes:` (page 784)
- `addIndexesInRange:` (page 785)

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 iOS 2.0 and later.

See Also
- `addIndex:` (page 784)
- `addIndexesInRange:` (page 785)

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` (page 1773) when *indexRange* would add an index that exceeds the maximum allowed value for unsigned integers.

Availability
Available in iOS 2.0 and later.

See Also
- `addIndex:` (page 784)
- `addIndexes:` (page 784)

Declared In
`NSIndexSet.h`

removeAllIndexes

Removes the receiver's indexes.

```
- (void)removeAllIndexes
```

Availability
Available in iOS 2.0 and later.

See Also
- `removeIndex:` (page 786)
- `removeIndexes:` (page 786)
- `removeIndexesInRange:` (page 786)

Declared In
`NSIndexSet.h`

removeIndex:

Removes an index from the receiver.

```
- (void)removeIndex:(NSUInteger)index
```

Parameters

index

Index to remove.

Availability

Available in iOS 2.0 and later.

See Also

- `removeAllIndexes` (page 785)
- `removeIndexes:` (page 786)
- `removeIndexesInRange:` (page 786)

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 iOS 2.0 and later.

See Also

- `removeIndex:` (page 786)
- `removeAllIndexes` (page 785)
- `removeIndexesInRange:` (page 786)

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 iOS 2.0 and later.

See Also
- `removeIndex:` (page 786)
- `removeIndexes:` (page 786)
- `removeAllIndexes` (page 785)

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 iOS 2.0 and later.

Declared In
`NSIndexSet.h`

NSMutableSet Class Reference

Inherits from	NSSet : NSObject
Conforms to	NSCoding (NSSet) NSCopying (NSSet) NSMutableCopying (NSSet) NSFastEnumeration (NSSet) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSSet.h
Companion guide	Collections Programming Topics
Related sample code	CryptoExercise ScrollViewSuite

Overview

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 791), which adds a single object to the set; `addObjectsFromArray:` (page 792), which adds all objects from a specified array to the set; or `unionSet:` (page 795), which adds all the objects from another set. You remove objects from an `NSMutableSet` object using any of the methods `intersectSet:` (page 793), `minusSet:` (page 794), `removeAllObjects` (page 794), or `removeObject:` (page 794).

When an object is added to a set, it receives a `retain` (page 1638) message. When an object is removed from a mutable set, it receives a `release` (page 1636) 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 1638) 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 791)
Creates and returns a mutable set with a given initial capacity.

`- initWithCapacity:` (page 793)
Returns an initialized mutable set with a given initial capacity.

Adding and Removing Entries

`- addObject:` (page 791)
Adds a given object to the receiver, if it is not already a member.

`- filterUsingPredicate:` (page 792)
Evaluates a given predicate against the receiver's content and removes from the receiver those objects for which the predicate returns false.

`- removeObject:` (page 794)
Removes a given object from the receiver.

`- removeAllObjects` (page 794)
Empties the receiver of all of its members.

`- addObjectsFromArray:` (page 792)
Adds to the receiver each object contained in a given array that is not already a member.

Combining and Recombining Sets

`- unionSet:` (page 795)
Adds to the receiver each object contained in another given set that is not already a member.

`- minusSet:` (page 794)
Removes from the receiver each object contained in another given set that is present in the receiver.

`- intersectSet:` (page 793)
Removes from the receiver each object that isn't a member of another given set.

`- setSet:` (page 795)
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 iOS 2.0 and later.

See Also

- `initWithCapacity:` (page 793)
+ `set` (page 1137) (`NSSet`)
+ `setWithObjects:count:` (page 1139) (`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 iOS 2.0 and later.

See Also

- `addObjectsFromArray:` (page 792)

- unionSet: (page 795)

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

See Also
- addObject: (page 791)
- unionSet: (page 795)

Declared In
NSSet.h

filterUsingPredicate:

Evaluates a given predicate against the receiver's content and removes from the receiver those objects for which the predicate returns false.

```
- (void)filterUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate
A predicate.

Discussion
The following example illustrates the use of this method.

```
NSMutableSet *mutableSet =
    [NSMutableSet setWithObjects:@"One", @"Two", @"Three", @"Four", nil];
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith 'T'"];
[mutableSet filterUsingPredicate:predicate];
// mutableSet contains (Two, Three)
```

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.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 iOS 2.0 and later.

See Also
+ setWithCapacity: (page 791)

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 iOS 2.0 and later.

See Also
- removeObject: (page 794)
- removeAllObjects (page 794)
- minusSet: (page 794)

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 iOS 2.0 and later.

See Also

- `removeObject:` (page 794)
- `removeAllObjects` (page 794)
- `intersectSet:` (page 793)

Declared In

`NSSet.h`

removeAllObjects

Empties the receiver of all of its members.

```
- (void)removeAllObjects
```

Availability

Available in iOS 2.0 and later.

See Also

- `removeObject:` (page 794)
- `minusSet:` (page 794)
- `intersectSet:` (page 793)

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 iOS 2.0 and later.

See Also
- `removeAllObjects` (page 794)
- `minusSet:` (page 794)
- `intersectSet:` (page 793)

Related Sample Code
ScrollViewSuite

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 iOS 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 iOS 2.0 and later.

See Also
- `addObject:` (page 791)
- `addObjectsFromArray:` (page 792)

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 iOS 2.0 and later.
Declared in	Foundation/NSString.h
Companion guide	String Programming Guide
Related sample code	KeyboardAccessory

Overview

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 801)—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 800) simply replaces the characters in a range of `0` length, while `deleteCharactersInRange:` (page 800) replaces the characters in a given range with no characters.

Tasks

Creating and Initializing a Mutable String

`+ stringWithCapacity:` (page 798)
Returns an empty `NSMutableString` object with initial storage for a given number of characters.

`- initWithCapacity:` (page 800)
Returns an `NSMutableString` object initialized with initial storage for a given number of characters,

Modifying a String

- `appendFormat:` (page 799)
  Adds a constructed string to the receiver.
- `appendString:` (page 799)
  Adds to the end of the receiver the characters of a given string.
- `deleteCharactersInRange:` (page 800)
  Removes from the receiver the characters in a given range.
- `insertString:atIndex:` (page 800)
  Inserts into the receiver the characters of a given string at a given location.
- `replaceCharactersInRange:withString:` (page 801)
  Replaces the characters from *aRange* with those in *aString*.
- `replaceOccurrencesOfString:withString:options:range:` (page 802)
  Replaces all occurrences of a given string in a given range with another given string, returning the number of replacements.
- `setString:` (page 802)
  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 iOS 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 1203) method with the arguments listed.

Availability

Available in iOS 2.0 and later.

See Also

`- appendString:` (page 799)

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 iOS 2.0 and later.

See Also

`- appendFormat:` (page 799)

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code

KeyboardAccessory

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 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 iOS 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 iOS 2.0 and later.

Declared In

`NSString.h`

NSMutableURLRequest Class Reference

Inherits from	NSURLRequest : NSObject
Conforms to	NSCoding (NSURLRequest) NSCopying (NSURLRequest) NSMutableCopying (NSURLRequest) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLRequest.h
Companion guide	URL Loading System Programming Guide

Overview

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 1462) and `setProperty:forKey:inRequest:` (page 1464) methods.

Tasks

Setting Request Properties

- `setCachePolicy:` (page 807)
  Sets the cache policy of the receiver.
- `setMainDocumentURL:` (page 809)
  Sets the main document URL for the receiver.

- `setNetworkServiceType:` (page 810)
  Sets the network service type of the connection.
- `setTimeoutInterval:` (page 810)
  Sets the receiver's timeout interval, in seconds.
- `setURL:` (page 810)
  Sets the URL of the receiver

Setting HTTP Specific Properties

- `addValue:forHTTPHeaderField:` (page 806)
  Adds an HTTP header to the receiver's HTTP header dictionary.
- `setAllHTTPHeaderFields:` (page 807)
  Replaces the receiver's header fields with the passed values.
- `setHTTPBody:` (page 808)
  Sets the request body of the receiver to the specified data.
- `setHTTPBodyStream:` (page 808)
  Sets the request body of the receiver to the contents of a specified input stream.
- `setHTTPMethod:` (page 808)
  Sets the receiver's HTTP request method.
- `setHTTPShouldHandleCookies:` (page 809)
  Sets whether the receiver should use the default cookie handling for the request.
- `setHTTPShouldUsePipelining:` (page 809)
  Sets whether the request can continue transmitting data before receiving a response from an earlier transmission.
- `setValue:forHTTPHeaderField:` (page 811)
  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

See Also
- `setValue:forHTTPHeaderField:` (page 811)

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

See Also
- `setValue:forHTTPHeaderField:` (page 811)

Declared In
`NSURLRequest.h`

setCachePolicy:

Sets the cache policy of the receiver.

```
- (void)setCachePolicy:(NSURLRequestCachePolicy)policy
```

Parameters

policy
The new cache policy.

Availability

See Also
- `cachePolicy` (page 1472)

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 808). These values are mutually exclusive.

Availability

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 808). These values are mutually exclusive.

Availability

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

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

Declared In
NSURLRequest.h

setHTTPShouldUsePipelining:

Sets whether the request can continue transmitting data before receiving a response from an earlier transmission.

```
- (void)setHTTPShouldUsePipelining:(BOOL)shouldUsePipelining
```

Parameters

shouldUsePipelining

If YES, the request should continue transmitting data; if NO, the request should wait for a response.

Discussion
Specifying YES does not guarantee HTTP pipelining behavior, because some servers do not support pipelining.

Availability
Available in iOS 4.0 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

Declared In
NSURLRequest.h

setNetworkServiceType:

Sets the network service type of the connection.

```
- (void)setNetworkServiceType:(NSURLRequestNetworkServiceType)networkServiceType
```

Parameters

networkServiceType
The network service type.

Availability
Available in iOS 4.0 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

See Also
- timeoutInterval (page 1476)

Declared In
NSURLRequest.h

setURL:

Sets the URL of the receiver

```
- (void)setURL:(NSURL *)theURL
```

Parameters

theURL
The new URL.

Availability

See Also
- URL (page 1476)

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

See Also
- addValue:forHTTPHeaderField: (page 806)

Declared In
NSURLRequest.h

NSNetService Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	BonjourWeb CryptoExercise WiTap

Overview

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 821) 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 823) method to verify that the service is available and ready for your application. If it is, the `addresses` (page 817) 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 819)
  Returns the receiver, initialized as a network service of a given type and sets the initial host information.
- `initWithDomain:type:name:port:` (page 819)
  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 815)
  Returns an `NSData` object representing a TXT record formed from a given dictionary.
+ `dictionaryFromTXTRecordData:` (page 816)
  Returns a dictionary representing a TXT record given as an `NSData` object.
- `addresses` (page 817)
  Returns an array containing `NSData` objects, each of which contains a socket address for the service.
- `domain` (page 817)
  Returns the domain name of the service.
- `getInputStream:outputStream:` (page 818)
  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 818)
  Returns the host name of the computer providing the service.
- `name` (page 821)
  Returns the name of the service.
- `type` (page 826)
  Returns the type of the service.
- `TXTRecordData` (page 826)
  Returns the TXT record for the receiver.

- `setTXTRecordData:` (page 824)
  Sets the TXT record for the receiver, and returns a Boolean value that indicates whether the operation was successful.
- `delegate` (page 817)
  Returns the delegate for the receiver.
- `setDelegate:` (page 824)
  Sets the delegate for the receiver.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 823)
  Adds the service to the specified run loop.
- `removeFromRunLoop:forMode:` (page 822)
  Removes the service from the given run loop for a given mode.

Using Network Services

- `publish` (page 821)
  Attempts to advertise the receiver's on the network.
- `publishWithOptions:` (page 822)
  Attempts to advertise the receiver on the network, with the given options.
- `resolve` (page 822)
  Starts a resolve process for the receiver. (Deprecated. Use `resolveWithTimeout:` (page 823) instead.)
- `resolveWithTimeout:` (page 823)
  Starts a resolve process of a finite duration for the receiver.
- `port` (page 821)
  Provides the port of the receiver.
- `startMonitoring` (page 825)
  Starts the monitoring of TXT-record updates for the receiver.
- `stop` (page 825)
  Halts a currently running attempt to publish or resolve a service.
- `stopMonitoring` (page 825)
  Stops the monitoring of TXT-record updates for the receiver.

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*. Fails an assertion if *txtDictionary* cannot be represented as an NSData object.

Availability

Available in iOS 2.0 and later.

See Also

- TXTRecordData (page 826)
+ dictionaryFromTXTRecordData: (page 816)

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.

Fails an assertion if *txtData* cannot be represented as an NSDictionary object.

Availability

Available in iOS 2.0 and later.

See Also

- TXTRecordData (page 826)
+ dataFromTXTRecordDictionary: (page 815)

Related Sample Code

BonjourWeb

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 iOS 2.0 and later.

See Also
`- resolve` (page 822)

Declared In
`NSNetServices.h`

delegate

Returns the delegate for the receiver.

```
- (id < NSNetServiceDelegate >)delegate
```

Return Value
The delegate for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 824)

Related Sample Code
CryptoExercise

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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 819).

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 823) 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 821) on an NSNetService object initialized with this method generates a call to your delegate's netService:didNotPublish: (page 1622) method with an NSNetServicesBadArgumentError (page 828) error.

Availability
Available in iOS 2.0 and later.

See Also
- initWithDomain:type:name:port: (page 819)

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 819) 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 iOS 2.0 and later.

See Also

- initWithDomain:type:name: (page 819)

Related Sample Code

CryptoExercise

WiTap

Declared In

NSNetServices.h

name

Returns the name of the service.

```
- (NSString *)name
```

Return Value
The name of the service.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
WiTap

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 iOS 2.0 and later.

See Also
`- stop` (page 825)

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 iOS 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 823) 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 iOS 2.0 and later.

See Also

- scheduleInRunLoop:forMode: (page 823)

Declared In

NSNetServices.h

resolve

Starts a resolve process for the receiver. (Deprecated in iOS 2.0. Use resolveWithTimeout: (page 823) 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 823) with a timeout value of `5`.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 2.0.

See Also
- `addresses` (page 817)
- `stop` (page 825)
- `resolveWithTimeout:` (page 823)

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 1623) to the delegate. Otherwise, the receiver sends `netService:didNotResolve:` (page 1622) to the delegate.

Availability
Available in iOS 2.0 and later.

See Also
- `addresses` (page 817)
- `stop` (page 825)

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 822) to transfer a service to a different run loop. You should not attempt to run a service on multiple run loops.

Availability
Available in iOS 2.0 and later.

See Also
- removeFromRunLoop:forMode: (page 822)

Declared In
NSNetServices.h

setDelegate:

Sets the delegate for the receiver.

```
- (void)setDelegate:(id < NSNetServiceDelegate >)delegate
```

Parameters
delegate
The delegate for the receiver. The delegate must conform to the NSNetServiceDelegate Protocol protocol.

Discussion
The delegate is not retained.

Availability
Available in iOS 2.0 and later.

See Also
- delegate (page 817)

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 iOS 2.0 and later.

See Also
- `TXTRecordData` (page 826)

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 1623), which is called when the TXT record for the receiver is updated.

Availability
Available in iOS 2.0 and later.

See Also
- `stopMonitoring` (page 825)

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 1624) message to the delegate.

Availability
Available in iOS 2.0 and later.

Declared In
`NSNetServices.h`

stopMonitoring

Stops the monitoring of TXT-record updates for the receiver.

```
- (void)stopMonitoring
```

Availability
Available in iOS 2.0 and later.

See Also
- `startMonitoring` (page 825)

Declared In
`NSNetServices.h`

TXTRecordData

Returns the TXT record for the receiver.

```
- (NSData *)TXTRecordData
```

Availability
Available in iOS 2.0 and later.

See Also
- `setTXTRecordData:` (page 824)
+ `dictionaryFromTXTRecordData:` (page 816)
+ `dataFromTXTRecordDictionary:` (page 815)

Related Sample Code
BonjourWeb

Declared In
`NSNetServices.h`

type

Returns the type of the service.

```
- (NSString *)type
```

Return Value
The type of the service.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesNotFoundError`

The service could not be found on the network.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesBadArgumentError`
An invalid argument was used when creating the `NSNetService` object.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesCancelledError`
The client canceled the action.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesInvalidError`
The net service was improperly configured.

Available in iOS 2.0 and later.

Declared in `NSNetServices.h`.

`NSNetServicesTimeoutError`
The net service has timed out.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSNetServices.h`.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	BonjourWeb CryptoExercise WiTap

Overview

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 834) 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 832) and `removeFromRunLoop:forMode:` (page 831) 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 833) or `searchForBrowsableDomains` (page 832) methods. From the list of returned domains, you can pick one and use the `searchForServicesOfType:inDomain:` (page 833) 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 833) 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 831)
  Initializes an allocated `NSNetServiceBrowser` (page 829) object.

Configuring Network Service Browsers

- `delegate` (page 831)
  Returns the receiver's delegate.
- `setDelegate:` (page 834)
  Sets the receiver's delegate.

Using Network Service Browsers

- `searchForBrowsableDomains` (page 832)
  Initiates a search for domains visible to the host. This method returns immediately.
- `searchForRegistrationDomains` (page 833)
  Initiates a search for domains in which the host may register services.
- `searchForServicesOfType:inDomain:` (page 833)
  Starts a search for services of a particular type within a specific domain.
- `stop` (page 834)
  Halts a currently running search or resolution.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 832)
  Adds the receiver to the specified run loop.
- `removeFromRunLoop:forMode:` (page 831)
  Removes the receiver from the specified run loop.

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id < NSNetServiceBrowserDelegate >)delegate
```

Return Value
Delegate for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 834)

Related Sample Code
BonjourWeb
CryptoExercise
WiTap

Declared In
`NSNetServices.h`

init

Initializes an allocated `NSNetServiceBrowser` (page 829) object.

```
- (id)init
```

Return Value
Initialized `NSNetServiceBrowser` (page 829) object.

Availability
Available in iOS 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` (page 1113). See the `Run Loop Modes` (page 1113) section of the `NSRunLoop` class for other run loop mode values.

Discussion
You can use this method in conjunction with `scheduleInRunLoop:forMode:` (page 832) 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 iOS 2.0 and later.

See Also
`- scheduleInRunLoop:forMode:` (page 832)

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` (page 1113). See the `Run Loop Modes` (page 1113) section of the `NSRunLoop` class for other run loop mode values.

Discussion
You can use this method in conjunction with `removeFromRunLoop:forMode:` (page 831) 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 iOS 2.0 and later.

See Also
`- removeFromRunLoop:forMode:` (page 831)

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 1616) message for each domain discovered.

Availability
Available in iOS 2.0 and later.

See Also
`- searchForRegistrationDomains` (page 833)

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 1619) message to the delegate if the network was ready to initiate the search. The delegate receives a subsequent `netServiceBrowser:didFindDomain:moreComing:` (page 1616) 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 iOS 2.0 and later.

See Also
- `searchForBrowsableDomains` (page 832)
- `searchForServicesOfType:inDomain:` (page 833)
- `netServiceBrowser:didFindDomain:moreComing:` (page 1616) (NSNetServerBrowserDelegate)
- `netServiceBrowserWillSearch:` (page 1619) (NSNetServerBrowserDelegate)

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 1619) message to the delegate if the network was ready to initiate the search.The delegate receives subsequent `netServiceBrowser:didFindService:moreComing:` (page 1616) 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 domains. 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 iOS 2.0 and later.

See Also
- `netServiceBrowser:didFindDomain:moreComing:` (page 1616) (NSNetServiceBrowserDelegate)
- `netServiceBrowserWillSearch:` (page 1619) (NSNetServiceBrowserDelegate)

Declared In
`NSNetServices.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id < NSNetServiceBrowserDelegate >)delegate
```

Parameters
delegate
Object to serve as the receiver's delegate. Must not be `nil`. The delegate must conform to the `NSNetServiceBrowserDelegate Protocol` protocol.

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 iOS 2.0 and later.

See Also
- `delegate` (page 831)

Declared In
`NSNetServices.h`

stop

Halts a currently running search or resolution.

```
- (void)stop
```

Discussion
This method sends a `netServiceBrowserDidStopSearch:` (page 1618) message to the delegate and causes the browser to discard any pending search results.

Availability
Available in iOS 2.0 and later.

See Also
`- netServiceBrowserDidStopSearch:` (page 1618) (NSNetServiceBrowserDelegate)

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 iOS 2.0 and later.
Declared in	Foundation/NSNotification.h
Companion guide	Notification Programming Topics
Related sample code	GKRocket KeyboardAccessory MoviePlayer SpeakHere

Overview

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 839) or `notificationWithName:object:userInfo:` (page 839). However, you don't usually create your own notifications directly. The NSNotificationCenter methods `postNotificationName:object:` (page 849) and `postNotificationName:object:userInfo:` (page 849) 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 840), `object` (page 840), and `userInfo` (page 841). 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 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Creating Notifications

`+ notificationWithName:object:` (page 839)
Returns a new notification object with a specified name and object.

`+ notificationWithName:object:userInfo:` (page 839)
Returns a notification object with a specified name, object, and user information.

Getting Notification Information

`- name` (page 840)
Returns the name of the notification.

`- object` (page 840)
Returns the object associated with the notification.

`- userInfo` (page 841)
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 iOS 2.0 and later.

See Also
`- postNotificationName:object:` (page 849) (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 iOS 2.0 and later.

See Also
`+ notificationWithName:object:` (page 839)
`- postNotificationName:object:userInfo:` (page 849) (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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
KeyboardAccessory

Declared In
`NSNotification.h`

NSNotificationCenter Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSNotification.h
Companion guide	Notification Programming Topics
Related sample code	AddMusic GKRocket KeyboardAccessory MoviePlayer SpeakHere

Overview

An `NSNotificationCenter` object (or simply, **notification center**) provides a mechanism for broadcasting information within a program. 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 846) or `addObserverForName:object:queue:usingBlock:` (page 847) methods. Each invocation of this method specifies a set of notifications. Therefore, objects may register as observers of different notification sets by calling these methods 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.

A notification center maintains a **notification dispatch table** which 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 61-1 shows the four types of dispatch table entries and the notification sets they specify. (This table omits the always present notification observer.)

Table 61-1 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 61-2 shows an example dispatch table with four observers.

Table 61-2 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 61-2 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.

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 848) 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 850) or `removeObserver:name:object:` (page 851)) 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 running Cocoa program has a default notification center. You typically don't create your own. An `NSNotificationCenter` object can deliver notifications only within a single program. If you want to post a notification to other processes or receive notifications from other processes, use a `NSDistributedNotificationCenter` object.

Tasks

Getting the Notification Center

`+ defaultCenter` (page 846)
Returns the process's default notification center.

Managing Notification Observers

`- addObserver:selector:name:object:` (page 846)
Adds an entry to the receiver's dispatch table with an observer, a notification selector and optional criteria: notification name and sender.

`- addObserverForName:object:queue:usingBlock:` (page 847)
Adds an entry to the receiver's dispatch table with a notification queue and a block to add to the queue, and optional criteria: notification name and sender.

`- removeObserver:` (page 850)
Removes all the entries specifying a given observer from the receiver's dispatch table.

`- removeObserver:name:object:` (page 851)
Removes matching entries from the receiver's dispatch table.

Posting Notifications

`- postNotification:` (page 848)
Posts a given notification to the receiver.

`- postNotificationName:object:` (page 849)
Creates a notification with a given name and sender and posts it to the receiver.

`- postNotificationName:object:userInfo:` (page 849)
Creates a notification with a given name, sender, and information and posts it to the receiver.

Class Methods

defaultCenter

Returns the process's default notification center.

```
+ (id)defaultCenter
```

Return Value
The current process's default notification center, which is used for system notifications.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
GKRocket
KeyboardAccessory
MoviePlayer
SpeakHere

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. This value must not be `nil`.

notificationSelector

Selector that specifies the message the receiver sends *notificationObserver* to notify it of the notification posting. The method specified by *notificationSelector* must have one and only one argument (an instance of `NSNotification`).

notificationName

The name of the notification for which to register the observer; that is, only notifications with this name are delivered to the observer.

If you pass `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.

If you pass `nil`, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer.

Discussion

Be sure to invoke `removeObserver:` (page 850) or `removeObserver:name:object:` (page 851) before *notificationObserver* or any object specified in `addObserver:selector:name:object:` is deallocated.

Availability

Available in iOS 2.0 and later.

See Also

- `addObserverForName:object:queue:usingBlock:` (page 847)
- `removeObserver:` (page 850)

Related Sample Code

AddMusic
GKRocket
KeyboardAccessory
MoviePlayer
SpeakHere

Declared In

`NSNotification.h`

addObserverForName:object:queue:usingBlock:

Adds an entry to the receiver's dispatch table with a notification queue and a block to add to the queue, and optional criteria: notification name and sender.

```
- (id)addObserverForName:(NSString *)name
    object:(id)obj
    queue:(NSOperationQueue *)queue
    usingBlock:(void (^)(NSNotification *))block
```

Parameters

name

The name of the notification for which to register the observer; that is, only notifications with this name are used to add the block to the operation queue.

If you pass `nil`, the notification center doesn't use a notification's name to decide whether to add the block to the operation queue.

obj

The object whose notifications you want to add the block to the operation queue.

If you pass `nil`, the notification center doesn't use a notification's sender to decide whether to add the block to the operation queue.

queue

The operation queue to which *block* should be added.

If you pass `nil`, the block is run synchronously on the posting thread.

block

The block to be executed when the notification is received.

The block is copied by the notification center and (the copy) held until the observer registration is removed.

The block takes one argument:

notification

The notification.

Return Value
An opaque object to act as the observer.

Discussion
To unregister observations, you pass the object returned by this method to `removeObserver:` (page 850). You must invoke `removeObserver:` (page 850) or `removeObserver:name:object:` (page 851) before any object specified by `addObserverForName:object:queue:usingBlock:` is deallocated.

If a given notification triggers more than one observer block, the blocks may all be executed concurrently with respect to one another (but on their given queue or on the current thread).

Special Considerations

In a garbage collected environment, the system does not keep a reference to observers, and registrations are automatically cleaned up when an observer is collected. You therefore need to ensure that the observer object is not collected for as long as you want the notification registration to remain. You must either:

1. Maintain a strong reference to the returned observer object somewhere (for example in an instance variable or in a global variable).

   You typically do this if you intend to explicitly call `removeObserver:` (page 850) on it at some point.

2. Retain the object (using `CFRetain`).

   You would do this if you intend to never remove the observer.

(In a reference counted environment, the system retains the returned observer object until it is removed, so there is no need to retain it yourself.)

Availability
Available in iOS 4.0 and later.

See Also
- `addObserver:selector:name:object:` (page 846)
- `removeObserver:` (page 850)

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 839) or `notificationWithName:object:userInfo:` (page 839). An exception is raised if *notification* is `nil`.

Availability

Available in iOS 2.0 and later.

See Also

- `postNotificationName:object:` (page 849)
- `postNotificationName:object:userInfo:` (page 849)

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 849) with a *userInfo* argument of `nil`.

Availability

Available in iOS 2.0 and later.

See Also

- `postNotification:` (page 848)

Related Sample Code

MoviePlayer

SpeakHere

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 iOS 2.0 and later.

See Also

`- postNotificationName:object:` (page 849)

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 851)) before *notificationObserver* or any object specified in `addObserver:selector:name:object:` (page 846) 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 iOS 2.0 and later.

See Also

`- removeObserver:name:object:` (page 851)

Related Sample Code

GKRocket

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 850)) before the observer object or any object specified in `addObserver:selector:name:object:` (page 846) is deallocated.

Availability

Available in iOS 2.0 and later.

See Also

`- removeObserver:` (page 850)

Related Sample Code

AddMusic

KeyboardAccessory

MoviePlayer

Declared In

`NSNotification.h`

NSNotificationQueue Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSNotificationQueue.h
Companion guide	Notification Programming Topics

Overview

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 856)
Initializes and returns a notification queue for the specified notification center.

Getting the Default Queue

`+ defaultQueue` (page 854)
Returns the default notification queue for the current thread.

Managing Notifications

- `enqueueNotification:postingStyle:` (page 855)
  Adds a notification to the notification queue with a specified posting style.
- `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855)
  Adds a notification to the notification queue with a specified posting style, criteria for coalescing, and runloop mode.
- `dequeueNotificationsMatching:coalesceMask:` (page 854)
  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 iOS 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 iOS 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 855).

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

`NSNotificationCoalescingOnName`
Coalesce notifications with the same name.

Available in iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

`NSNotificationCoalescingOnSender`
Coalesce notifications with the same object.

Available in iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

Discussion
These constants are used in the third argument of `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855). You can OR them together to specify more than one.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

`NSPostWhenIdle`
The notification is posted when the run loop is idle.

Available in iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

`NSPostNow`
The notification is posted immediately after coalescing.

Available in iOS 2.0 and later.

Declared in `NSNotificationQueue.h`.

Discussion
These constants are used in both `enqueueNotification:postingStyle:` (page 855) and `enqueueNotification:postingStyle:coalesceMask:forModes:` (page 855).

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSNull.h
Companion guide	Number and Value Programming Topics

Overview

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 1552)
- `- initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Obtaining an Instance

`+ null` (page 860)
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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSValue.h Foundation/NSDecimalNumber.h
Companion guides	Number and Value Programming Topics Property List Programming Guide
Related sample code	aurioTouch CryptoExercise GLSprite MoviePlayer SpeakHere

Overview

`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 871) 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 865)
Creates and returns an `NSNumber` object containing a given value, treating it as a `BOOL`.

`+ numberWithChar:` (page 865)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `char`.

`+ numberWithDouble:` (page 865)
Creates and returns an `NSNumber` object containing a given value, treating it as a `double`.

`+ numberWithFloat:` (page 866)
Creates and returns an `NSNumber` object containing a given value, treating it as a `float`.

`+ numberWithInt:` (page 866)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `int`.

`+ numberWithInteger:` (page 867)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSInteger`.

`+ numberWithLong:` (page 867)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long`.

`+ numberWithLongLong:` (page 867)
Creates and returns an `NSNumber` object containing a given value, treating it as a signed `long long`.

`+ numberWithShort:` (page 868)
Creates and returns an `NSNumber` object containing *`value`*, treating it as a signed `short`.

`+ numberWithUnsignedChar:` (page 868)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned char`.

`+ numberWithUnsignedInt:` (page 868)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned int`.

`+ numberWithUnsignedInteger:` (page 869)
Creates and returns an `NSNumber` object containing a given value, treating it as an `NSUInteger`.

`+ numberWithUnsignedLong:` (page 869)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long`.

`+ numberWithUnsignedLongLong:` (page 870)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned long long`.

`+ numberWithUnsignedShort:` (page 870)
Creates and returns an `NSNumber` object containing a given value, treating it as an `unsigned short`.

Initializing an NSNumber Object

`- initWithBool:` (page 874)
Returns an `NSNumber` object initialized to contain a given value, treated as a `BOOL`.

`- initWithChar:` (page 874)
Returns an `NSNumber` object initialized to contain a given value, treated as a signed `char`.

- `initWithDouble:` (page 874)
  Returns an `NSNumber` object initialized to contain *`value`*, treated as a `double`.
- `initWithFloat:` (page 875)
  Returns an `NSNumber` object initialized to contain a given value, treated as a `float`.
- `initWithInt:` (page 875)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `int`.
- `initWithInteger:` (page 875)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `NSInteger`.
- `initWithLong:` (page 876)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `long`.
- `initWithLongLong:` (page 876)
  Returns an `NSNumber` object initialized to contain *`value`*, treated as a signed `long long`.
- `initWithShort:` (page 876)
  Returns an `NSNumber` object initialized to contain a given value, treated as a signed `short`.
- `initWithUnsignedChar:` (page 877)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned char`.
- `initWithUnsignedInt:` (page 877)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned int`.
- `initWithUnsignedInteger:` (page 878)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `NSUInteger`.
- `initWithUnsignedLong:` (page 878)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long`.
- `initWithUnsignedLongLong:` (page 878)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned long long`.
- `initWithUnsignedShort:` (page 879)
  Returns an `NSNumber` object initialized to contain a given value, treated as an `unsigned short`.

Accessing Numeric Values

- `boolValue` (page 870)
  Returns the receiver's value as a `BOOL`.
- `charValue` (page 871)
  Returns the receiver's value as a `char`.
- `decimalValue` (page 872)
  Returns the receiver's value, expressed as an `NSDecimal` structure.
- `doubleValue` (page 873)
  Returns the receiver's value as a `double`.
- `floatValue` (page 873)
  Returns the receiver's value as a `float`.
- `intValue` (page 879)
  Returns the receiver's value as an `int`.
- `integerValue` (page 879)
  Returns the receiver's value as an `NSInteger`.

- `longLongValue` (page 880)
  Returns the receiver's value as a `long long`.
- `longValue` (page 880)
  Returns the receiver's value as a `long`.
- `shortValue` (page 881)
  Returns the receiver's value as a `short`.
- `unsignedCharValue` (page 881)
  Returns the receiver's value as an unsigned `char`.
- `unsignedIntegerValue` (page 882)
  Returns the receiver's value as an `NSUInteger`.
- `unsignedIntValue` (page 882)
  Returns the receiver's value as an unsigned `int`.
- `unsignedLongLongValue` (page 882)
  Returns the receiver's value as an unsigned `long long`.
- `unsignedLongValue` (page 883)
  Returns the receiver's value as an unsigned `long`.
- `unsignedShortValue` (page 883)
  Returns the receiver's value as an unsigned `short`.

Retrieving String Representations

- `descriptionWithLocale:` (page 872)
  Returns a string that represents the contents of the receiver for a given locale.
- `stringValue` (page 881)
  Returns the receiver's value as a human-readable string.

Comparing NSNumber Objects

- `compare:` (page 871)
  Returns an `NSComparisonResult` value that indicates whether the receiver is greater than, equal to, or less than a given number.
- `isEqualToNumber:` (page 880)
  Returns a Boolean value that indicates whether the receiver and a given number are equal.

Accessing Type Information

- `objCType` (page 881)
  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 iOS 2.0 and later.

Related Sample Code

aurioTouch

CryptoExercise

GLSprite

SpeakHere

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code

SpeakHere

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 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 iOS 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 iOS 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 1240) 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 iOS 2.0 and later.

See Also
`- stringValue` (page 881)

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 `NO`.

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 871) method).

This method is more efficient than `compare:` (page 871) if you know the two objects are numbers.

Availability

Available in iOS 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 iOS 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 iOS 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 iOS 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 872) where locale is `nil`.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`NSValue.h`

NSNumberFormatter Class Reference

Inherits from	NSFormatter : NSObject
Conforms to	NSCoding (NSFormatter) NSCopying (NSFormatter) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSNumberFormatter.h
Companion guide	Data Formatting Guide

Overview

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 897) method and you can set the formatter behavior with the `setFormatterBehavior:` (page 915) method.

iOS Note: iOS supports only the modern 10.4+ behavior. 10.0-style methods and format strings are not available on iOS.

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`.

Nomenclature note: `NSNumberFormatter` provides several methods (such as `setMaximumFractionDigits:` (page 919)) 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 915)
  Sets the formatter behavior of the receiver.
- `formatterBehavior` (page 897)
  Returns an `NSNumberFormatterBehavior` constant that indicates the formatter behavior of the receiver.
+ `setDefaultFormatterBehavior:` (page 894)
  Sets the default formatter behavior for new instances of `NSNumberFormatter`.
+ `defaultFormatterBehavior` (page 893)
  Returns an `NSNumberFormatterBehavior` constant that indicates default formatter behavior for new instances of `NSNumberFormatter`.
- `setNumberStyle:` (page 925)
  Sets the number style used by the receiver.
- `numberStyle` (page 907)
  Returns the number-formatter style of the receiver.
- `setGeneratesDecimalNumbers:` (page 916)
  Controls whether the receiver creates instances of `NSDecimalNumber` when it converts strings to number objects.
- `generatesDecimalNumbers` (page 898)
  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 898)
  Returns by reference a cell-content object after creating it from a range of characters in a given string.

- `numberFromString:` (page 907)
  Returns an `NSNumber` object created by parsing a given string.
- `stringFromNumber:` (page 934)
  Returns a string containing the formatted value of the provided number object.
+ `localizedStringFromNumber:numberStyle:` (page 893)
  Returns a localized date string with the specified style.

Managing Localization of Numbers

- `setLocale:` (page 918)
  Sets the locale of the receiver.
- `locale` (page 901)
  Returns the locale of the receiver.

Configuring Rounding Behavior

- `setRoundingIncrement:` (page 929)
  Sets the rounding increment used by the receiver.
- `roundingIncrement` (page 911)
  Returns the rounding increment used by the receiver.
- `setRoundingMode:` (page 929)
  Sets the rounding mode used by the receiver.
- `roundingMode` (page 911)
  Returns the rounding mode used by the receiver.

Configuring Numeric Formats

- `setFormatWidth:` (page 916)
  Sets the format width used by the receiver.
- `formatWidth` (page 897)
  Returns the format width of the receiver.
- `setNegativeFormat:` (page 923)
  Sets the format the receiver uses to display negative values.
- `negativeFormat` (page 905)
  Returns the format used by the receiver to display negative numbers.
- `setPositiveFormat:` (page 928)
  Sets the format the receiver uses to display positive values.
- `positiveFormat` (page 910)
  Returns the format used by the receiver to display positive numbers.
- `setMultiplier:` (page 922)
  Sets the multiplier of the receiver.
- `multiplier` (page 905)
  Returns the multiplier used by the receiver as an `NSNumber` object.

Configuring Numeric Symbols

- `setPercentSymbol:` (page 926)
  Sets the string used by the receiver to represent the percent symbol.
- `percentSymbol` (page 909)
  Returns the string that the receiver uses to represent the percent symbol.
- `setPerMillSymbol:` (page 927)
  Sets the string used by the receiver to represent the per-mill (per-thousand) symbol.
- `perMillSymbol` (page 909)
  Returns the string that the receiver uses for the per-thousands symbol.
- `setMinusSign:` (page 922)
  Sets the string used by the receiver for the minus sign.
- `minusSign` (page 904)
  Returns the string the receiver uses to represent the minus sign.
- `setPlusSign:` (page 927)
  Sets the string used by the receiver to represent the plus sign.
- `plusSign` (page 909)
  Returns the string the receiver uses for the plus sign.
- `setExponentSymbol:` (page 915)
  Sets the string used by the receiver to represent the exponent symbol.
- `exponentSymbol` (page 897)
  Returns the string the receiver uses as an exponent symbol.
- `setZeroSymbol:` (page 934)
  Sets the string the receiver uses as the symbol to show the value zero.
- `zeroSymbol` (page 938)
  Returns the string the receiver uses as the symbol to show the value zero.
- `setNilSymbol:` (page 924)
  Sets the string the receiver uses to represent `nil` values.
- `nilSymbol` (page 906)
  Returns the string the receiver uses to represent a `nil` value.
- `setNotANumberSymbol:` (page 925)
  Sets the string the receiver uses to represent NaN (“not a number”).
- `notANumberSymbol` (page 907)
  Returns the symbol the receiver uses to represent NaN (“not a number”) when it converts values.
- `setNegativeInfinitySymbol:` (page 923)
  Sets the string used by the receiver for the negative infinity symbol.
- `negativeInfinitySymbol` (page 905)
  Returns the symbol the receiver uses to represent negative infinity.
- `setPositiveInfinitySymbol:` (page 928)
  Sets the string used by the receiver for the positive infinity symbol.
- `positiveInfinitySymbol` (page 910)
  Returns the string the receiver uses for the positive infinity symbol.

Configuring the Format of Currency

- `setCurrencySymbol:` (page 914)
  Sets the string used by the receiver as a local currency symbol.
- `currencySymbol` (page 896)
  Returns the receiver's local currency symbol.
- `setCurrencyCode:` (page 913)
  Sets the receiver's currency code.
- `currencyCode` (page 895)
  Returns the receiver's currency code as a string.
- `setInternationalCurrencySymbol:` (page 917)
  Sets the string used by the receiver for the international currency symbol.
- `internationalCurrencySymbol` (page 900)
  Returns the international currency symbol used by the receiver.
- `setCurrencyGroupingSeparator:` (page 914)
  Sets the currency grouping separator for the receiver.
- `currencyGroupingSeparator` (page 896)
  Returns the currency grouping separator for the receiver.

Configuring Numeric Prefixes and Suffixes

- `setPositivePrefix:` (page 928)
  Sets the string the receiver uses as the prefix for positive values.
- `positivePrefix` (page 910)
  Returns the string the receiver uses as the prefix for positive values.
- `setPositiveSuffix:` (page 929)
  Sets the string the receiver uses as the suffix for positive values.
- `positiveSuffix` (page 911)
  Returns the string the receiver uses as the suffix for positive values.
- `setNegativePrefix:` (page 923)
  Sets the string the receiver uses as a prefix for negative values.
- `negativePrefix` (page 906)
  Returns the string the receiver inserts as a prefix to negative values.
- `setNegativeSuffix:` (page 924)
  Sets the string the receiver uses as a suffix for negative values.
- `negativeSuffix` (page 906)
  Returns the string the receiver adds as a suffix to negative values.

Configuring the Display of Numeric Values

- `setTextAttributesForNegativeValues:` (page 931)
  Sets the text attributes to be used in displaying negative values .

- `textAttributesForNegativeValues` (page 935)
  Returns a dictionary containing the text attributes that have been set for negative values.
- `setTextAttributesForPositiveValues:` (page 932)
  Sets the text attributes to be used in displaying positive values.
- `textAttributesForPositiveValues` (page 937)
  Returns a dictionary containing the text attributes that have been set for positive values.
- `setTextAttributesForZero:` (page 933)
  Sets the text attributes used to display a zero value.
- `textAttributesForZero` (page 937)
  Returns a dictionary containing the text attributes used to display a value of zero.
- `setTextAttributesForNil:` (page 931)
  Sets the text attributes used to display the `nil` symbol.
- `textAttributesForNil` (page 936)
  Returns a dictionary containing the text attributes used to display the `nil` symbol.
- `setTextAttributesForNotANumber:` (page 932)
  Sets the text attributes used to display the NaN ("not a number") string.
- `textAttributesForNotANumber` (page 936)
  Returns a dictionary containing the text attributes used to display the NaN ("not a number") symbol.
- `setTextAttributesForPositiveInfinity:` (page 932)
  Sets the text attributes used to display the positive infinity symbol.
- `textAttributesForPositiveInfinity` (page 936)
  Returns a dictionary containing the text attributes used to display the positive infinity symbol.
- `setTextAttributesForNegativeInfinity:` (page 930)
  Sets the text attributes used to display the negative infinity symbol.
- `textAttributesForNegativeInfinity` (page 935)
  Returns a dictionary containing the text attributes used to display the negative infinity string.

Configuring Separators and Grouping Size

- `setGroupingSeparator:` (page 917)
  Specifies the string used by the receiver for a grouping separator.
- `groupingSeparator` (page 899)
  Returns a string containing the receiver's grouping separator.
- `setUsesGroupingSeparator:` (page 933)
  Controls whether the receiver displays the grouping separator.
- `usesGroupingSeparator` (page 937)
  Returns a Boolean value that indicates whether the receiver uses the grouping separator.
- `setDecimalSeparator:` (page 914)
  Sets the character the receiver uses as a decimal separator.
- `decimalSeparator` (page 896)
  Returns a string containing the character the receiver uses to represent decimal separators.
- `setAlwaysShowsDecimalSeparator:` (page 912)
  Controls whether the receiver always shows the decimal separator, even for integer numbers.

- `alwaysShowsDecimalSeparator` (page 894)
  Returns a Boolean value that indicates whether the receiver always shows a decimal separator, even if the number is an integer.
- `setCurrencyDecimalSeparator:` (page 913)
  Sets the string used by the receiver as a decimal separator.
- `currencyDecimalSeparator` (page 895)
  Returns the receiver's currency decimal separator as a string.
- `setGroupingSize:` (page 917)
  Sets the grouping size of the receiver.
- `groupingSize` (page 899)
  Returns the receiver's primary grouping size.
- `setSecondaryGroupingSize:` (page 930)
  Sets the secondary grouping size of the receiver.
- `secondaryGroupingSize` (page 912)
  Returns the size of secondary groupings for the receiver.

Managing the Padding of Numbers

- `setPaddingCharacter:` (page 925)
  Sets the string that the receiver uses to pad numbers in the formatted string representation.
- `paddingCharacter` (page 908)
  Returns a string containing the padding character for the receiver.
- `setPaddingPosition:` (page 926)
  Sets the padding position used by the receiver.
- `paddingPosition` (page 908)
  Returns the padding position of the receiver.

Managing Input Attributes

- `setAllowsFloats:` (page 912)
  Sets whether the receiver allows as input floating-point values (that is, values that include the period character [.]).
- `allowsFloats` (page 894)
  Returns a Boolean value that indicates whether the receiver allows floating-point values as input.
- `setMinimum:` (page 920)
  Sets the lowest number the receiver allows as input.
- `minimum` (page 903)
  Returns the lowest number allowed as input by the receiver.
- `setMaximum:` (page 919)
  Sets the highest number the receiver allows as input.
- `maximum` (page 901)
  Returns the highest number allowed as input by the receiver.
- `setMinimumIntegerDigits:` (page 921)
  Sets the minimum number of integer digits allowed as input by the receiver.

- `minimumIntegerDigits` (page 904)
  Returns the minimum number of integer digits allowed as input by the receiver.
- `setMinimumFractionDigits:` (page 921)
  Sets the minimum number of digits after the decimal separator allowed as input by the receiver.
- `minimumFractionDigits` (page 903)
  Returns the minimum number of digits after the decimal separator allowed as input by the receiver.
- `setMaximumIntegerDigits:` (page 919)
  Sets the maximum number of integer digits allowed as input by the receiver.
- `maximumIntegerDigits` (page 902)
  Returns the maximum number of integer digits allowed as input by the receiver.
- `setMaximumFractionDigits:` (page 919)
  Sets the maximum number of digits after the decimal separator allowed as input by the receiver.
- `maximumFractionDigits` (page 902)
  Returns the maximum number of digits after the decimal separator allowed as input by the receiver.

Configuring Significant Digits

- `setUsesSignificantDigits:` (page 934)
  Sets whether the receiver uses significant digits.
- `usesSignificantDigits` (page 938)
  Returns a Boolean value that indicates whether the receiver uses significant digits.
- `setMinimumSignificantDigits:` (page 921)
  Sets the minimum number of significant digits for the receiver.
- `minimumSignificantDigits` (page 904)
  Returns the minimum number of significant digits for the receiver.
- `setMaximumSignificantDigits:` (page 920)
  Sets the maximum number of significant digits for the receiver.
- `maximumSignificantDigits` (page 902)
  Returns the maximum number of significant digits for the receiver.

Managing Leniency Behavior

- `setLenient:` (page 918)
  Sets whether the receiver will use heuristics to guess at the number which is intended by a string.
- `isLenient` (page 900)
  Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the number which is intended by a string.

Managing the Validation of Partial Numeric Strings

- `setPartialStringValidationEnabled:` (page 926)
  Sets whether partial string validation is enabled for the receiver.

- isPartialStringValidationEnabled (page 900)
  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 iOS 2.0 and later.

See Also
+ setDefaultFormatterBehavior: (page 894)

Declared In
NSNumberFormatter.h

localizedStringFromNumber:numberStyle:

Returns a localized date string with the specified style.

```
+ (NSString *)localizedStringFromNumber:(NSNumber *)num
    numberStyle:(NSNumberFormatterStyle)localizationStyle
```

Parameters

num
The number to localize

localizationStyle
The localization style to use. See "NSNumberFormatterStyle" (page 939) for the supported values.

Return Value
An appropriately formatted NSString.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
`+ defaultFormatterBehavior` (page 893)

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 iOS 2.0 and later.

See Also
`- setAllowsFloats:` (page 912)

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 iOS 2.0 and later.

See Also
`- setAlwaysShowsDecimalSeparator:` (page 912)

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 iOS 2.0 and later.

See Also
`- setCurrencyCode:` (page 913)

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 iOS 2.0 and later.

See Also
`- setCurrencyDecimalSeparator:` (page 913)

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 iOS 2.0 and later.

See Also
- `setCurrencyGroupingSeparator:` (page 914)

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 iOS 2.0 and later.

See Also
- `internationalCurrencySymbol` (page 900)
- `setCurrencySymbol:` (page 914)

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 iOS 2.0 and later.

See Also
- `setDecimalSeparator:` (page 914)

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 iOS 2.0 and later.

See Also
- `setExponentSymbol:` (page 915)

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 iOS 2.0 and later.

See Also
- `setFormatterBehavior:` (page 915)

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 908).

Availability
Available in iOS 2.0 and later.

See Also
`- setFormatWidth:` (page 916)

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 iOS 2.0 and later.

See Also
`- setGeneratesDecimalNumbers:` (page 916)

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 898). 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 `NSControl` delegation method `control:didFailToFormatString:errorDescription:`.

Availability
Available in iOS 2.0 and later.

See Also
`- numberFromString:` (page 907)
`- stringFromNumber:` (page 934)

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 iOS 2.0 and later.

See Also
`- setGroupingSeparator:` (page 917)

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 iOS 2.0 and later.

See Also
- `setGroupingSize:` (page 917)

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 iOS 2.0 and later.

See Also
- `currencySymbol` (page 896)
- `setInternationalCurrencySymbol:` (page 917)

Declared In
`NSNumberFormatter.h`

isLenient

Returns a Boolean value that indicates whether the receiver uses heuristics to guess at the number which is intended by a string.

```
- (BOOL)isLenient
```

Return Value
`YES` if the receiver uses heuristics to guess at the number which is intended by the string; otherwise `NO`.

Availability
Available in iOS 2.0 and later.

See Also
- `setLenient:` (page 918)

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 iOS 2.0 and later.

See Also
`- setPartialStringValidationEnabled:` (page 926)

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 iOS 2.0 and later.

See Also
`- setLocale:` (page 918)

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

See Also
`- setMaximum:` (page 919)
`+ setDefaultFormatterBehavior:` (page 894)
`- formatterBehavior` (page 897)

- setFormatterBehavior: (page 915)

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 iOS 2.0 and later.

See Also
- setMaximumFractionDigits: (page 919)

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 iOS 2.0 and later.

See Also
- setMaximumIntegerDigits: (page 919)

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 iOS 2.0 and later.

See Also
- `setMaximumSignificantDigits:` (page 920)
- `minimumSignificantDigits` (page 904)
- `usesSignificantDigits` (page 938)

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

See Also
- `setMinimum:` (page 920)
- `+ setDefaultFormatterBehavior:` (page 894)
- `formatterBehavior` (page 897)
- `setFormatterBehavior:` (page 915)

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 iOS 2.0 and later.

See Also
- `setMinimumFractionDigits:` (page 921)

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 iOS 2.0 and later.

See Also
- `setMinimumIntegerDigits:` (page 921)

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 iOS 2.0 and later.

See Also
- `setMinimumSignificantDigits:` (page 921)
- `maximumSignificantDigits` (page 902)
- `usesSignificantDigits` (page 938)

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 iOS 2.0 and later.

See Also
- `setMinusSign:` (page 922)

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 iOS 2.0 and later.

See Also
- setMultiplier: (page 922)

Declared In
NSNumberFormatter.h

negativeFormat

Returns the format used by the receiver to display negative numbers.

```
- (NSString *)negativeFormat
```

Availability
Available in iOS 2.0 and later.

See Also
- setNegativeFormat: (page 923)

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 iOS 2.0 and later.

See Also
- `setNegativeInfinitySymbol:` (page 923)

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 iOS 2.0 and later.

See Also
- `negativeSuffix` (page 906)
- `setNegativePrefix:` (page 923)

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 iOS 2.0 and later.

See Also
- `negativePrefix` (page 906)
- `setNegativeSuffix:` (page 924)

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 iOS 2.0 and later.

See Also
- `setNilSymbol:` (page 924)

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 iOS 2.0 and later.

See Also
- `setNotANumberSymbol:` (page 925)

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 iOS 2.0 and later.

See Also
- `stringFromNumber:` (page 934)

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 iOS 2.0 and later.

See Also
`- setNumberStyle:` (page 925)

Declared In
`NSNumberFormatter.h`

paddingCharacter

Returns a string containing the padding character for the receiver.

```
- (NSString *)paddingCharacter
```

Availability
Available in iOS 2.0 and later.

See Also
`- setPaddingCharacter:` (page 925)

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 iOS 2.0 and later.

See Also
`- setPaddingPosition:` (page 926)

Declared In
`NSNumberFormatter.h`

percentSymbol

Returns the string that the receiver uses to represent the percent symbol.

```
- (NSString *)percentSymbol
```

Availability
Available in iOS 2.0 and later.

See Also
- setPercentSymbol: (page 926)

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 iOS 2.0 and later.

See Also
- setPerMillSymbol: (page 927)

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 iOS 2.0 and later.

See Also
- setPlusSign: (page 927)

Declared In
NSNumberFormatter.h

positiveFormat

Returns the format used by the receiver to display positive numbers.

```
- (NSString *)positiveFormat
```

Availability
Available in iOS 2.0 and later.

See Also
- setPositiveFormat: (page 928)

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 iOS 2.0 and later.

See Also
- setPositiveInfinitySymbol: (page 928)

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 iOS 2.0 and later.

See Also
- setPositivePrefix: (page 928)

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 iOS 2.0 and later.

See Also
- setPositiveSuffix: (page 929)

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 iOS 2.0 and later.

See Also
- setRoundingIncrement: (page 929)

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 iOS 2.0 and later.

See Also
- setRoundingMode: (page 929)

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 iOS 2.0 and later.

See Also
- `setSecondaryGroupingSize:` (page 930)

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 iOS 2.0 and later.

See Also
- `allowsFloats` (page 894)

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 iOS 2.0 and later.

See Also
- alwaysShowsDecimalSeparator (page 894)

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 iOS 2.0 and later.

See Also
- currencyCode (page 895)

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 iOS 2.0 and later.

See Also
- currencyDecimalSeparator (page 895)

Declared In
NSNumberFormatter.h

setCurrencyGroupingSeparator:

Sets the currency grouping separator for the receiver.

```
- (void)setCurrencyGroupingSeparator:(NSString *)string
```

Parameters

string

The currency grouping separator for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
- currencyGroupingSeparator (page 896)

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 iOS 2.0 and later.

See Also
- currencySymbol (page 896)
- setInternationalCurrencySymbol: (page 917)

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 iOS 2.0 and later.

See Also

- `decimalSeparator` (page 896)
- `formatterBehavior` (page 897)

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 iOS 2.0 and later.

See Also

- `exponentSymbol` (page 897)

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 iOS 2.0 and later.

See Also
- formatterBehavior (page 897)

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 908).

Availability
Available in iOS 2.0 and later.

See Also
- formatWidth (page 897)

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 iOS 2.0 and later.

See Also
- generatesDecimalNumbers (page 898)

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 iOS 2.0 and later.

See Also
- groupingSeparator (page 899)

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 iOS 2.0 and later.

See Also
- groupingSize (page 899)

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 iOS 2.0 and later.

See Also
- internationalCurrencySymbol (page 900)

Declared In
NSNumberFormatter.h

setLenient:

Sets whether the receiver will use heuristics to guess at the number which is intended by a string.

```
- (void)setLenient:(BOOL)b
```

Parameters

b

YES if the receiver will use heuristics to guess at the number 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 number wrong (that is, a number other than that which was intended).

Availability
Available in iOS 2.0 and later.

See Also
- isLenient (page 900)

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 iOS 2.0 and later.

See Also
- locale (page 901)

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

See Also

- `maximum` (page 901)
+ `setDefaultFormatterBehavior:` (page 894)
- `formatterBehavior` (page 897)
- `setFormatterBehavior:` (page 915)

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 iOS 2.0 and later.

See Also

- `maximumFractionDigits` (page 902)

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 iOS 2.0 and later.

See Also
- `minimumIntegerDigits` (page 904)

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 iOS 2.0 and later.

See Also
- `maximumSignificantDigits` (page 902)
- `setMinimumSignificantDigits:` (page 921)
- `usesSignificantDigits` (page 938)

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

See Also
- `minimum` (page 903)

+ setDefaultFormatterBehavior: (page 894)
- formatterBehavior (page 897)
- setFormatterBehavior: (page 915)

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 iOS 2.0 and later.

See Also
- minimumFractionDigits (page 903)

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 iOS 2.0 and later.

See Also
- minimumIntegerDigits (page 904)

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 iOS 2.0 and later.

See Also
- `minimumSignificantDigits` (page 904)
- `setMaximumSignificantDigits:` (page 920)
- `usesSignificantDigits` (page 938)

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 iOS 2.0 and later.

See Also
- `minusSign` (page 904)

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 iOS 2.0 and later.

See Also
- multiplier (page 905)

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 iOS 2.0 and later.

See Also
- negativeFormat (page 905)

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 iOS 2.0 and later.

See Also
- negativeInfinitySymbol (page 905)

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 iOS 2.0 and later.

See Also

- negativePrefix (page 906)

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 iOS 2.0 and later.

See Also

- negativeSuffix (page 906)
- negativePrefix (page 906)

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 iOS 2.0 and later.

See Also

- nilSymbol (page 906)

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 iOS 2.0 and later.

See Also
- notANumberSymbol (page 907)

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 iOS 2.0 and later.

See Also
- numberStyle (page 907)

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 iOS 2.0 and later.

See Also
- paddingCharacter (page 908)

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 iOS 2.0 and later.

See Also
- paddingPosition (page 908)

Declared In
NSNumberFormatter.h

setPartialStringValidationEnabled:

Sets whether partial string validation is enabled for the receiver.

```
- (void)setPartialStringValidationEnabled:(BOOL)enabled
```

Parameters

enabled
YES if partial string validation is enabled, otherwise NO.

Availability
Available in iOS 2.0 and later.

See Also
- isPartialStringValidationEnabled (page 900)

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 iOS 2.0 and later.

See Also

- percentSymbol (page 909)

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 iOS 2.0 and later.

See Also

- perMillSymbol (page 909)

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 iOS 2.0 and later.

See Also

- plusSign (page 909)

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 iOS 2.0 and later.

See Also
- positiveFormat (page 910)

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 iOS 2.0 and later.

See Also
- positiveInfinitySymbol (page 910)

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 iOS 2.0 and later.

See Also
- positivePrefix (page 910)

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 iOS 2.0 and later.

See Also
- positiveSuffix (page 911)

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 iOS 2.0 and later.

See Also
- roundingIncrement (page 911)

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 iOS 2.0 and later.

See Also
- `roundingMode` (page 911)

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 iOS 2.0 and later.

See Also
- `secondaryGroupingSize` (page 912)

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 iOS 2.0 and later.

See Also
- `textAttributesForNegativeInfinity` (page 935)
- `setNegativeInfinitySymbol:` (page 923)

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 iOS 2.0 and later.

See Also

- `textAttributesForNegativeValues` (page 935)

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 iOS 2.0 and later.

See Also

- `textAttributesForNil` (page 936)
- `nilSymbol` (page 906)

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 iOS 2.0 and later.

See Also
- setTextAttributesForNotANumber: (page 932)
- notANumberSymbol (page 907)

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 iOS 2.0 and later.

See Also
- positiveInfinitySymbol (page 910)
- textAttributesForPositiveInfinity (page 936)

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 931) for an example of how a related method might be used.

Availability
Available in iOS 2.0 and later.

See Also
- textAttributesForPositiveValues (page 937)

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 iOS 2.0 and later.

See Also
- textAttributesForZero (page 937)

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 iOS 2.0 and later.

See Also
- usesGroupingSeparator (page 937)

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 iOS 2.0 and later.

See Also

- `usesSignificantDigits` (page 938)
- `setMaximumSignificantDigits:` (page 920)
- `setMinimumSignificantDigits:` (page 921)

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 iOS 2.0 and later.

See Also

- `zeroSymbol` (page 938)

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 iOS 2.0 and later.

See Also

`- numberFromString:` (page 907)

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 iOS 2.0 and later.

See Also

`- setTextAttributesForNegativeInfinity:` (page 930)

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 iOS 2.0 and later.

See Also

`- setTextAttributesForNegativeValues:` (page 931)

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 iOS 2.0 and later.

See Also
`- setTextAttributesForNil:` (page 931)

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 iOS 2.0 and later.

See Also
`- setTextAttributesForNotANumber:` (page 932)
`- notANumberSymbol` (page 907)

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 iOS 2.0 and later.

See Also
`- setTextAttributesForPositiveInfinity:` (page 932)
`- positiveInfinitySymbol` (page 910)

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 iOS 2.0 and later.

See Also
- setTextAttributesForPositiveValues: (page 932)

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 iOS 2.0 and later.

See Also
- setTextAttributesForZero: (page 933)

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 iOS 2.0 and later.

See Also
- `setUsesGroupingSeparator:` (page 933)

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 iOS 2.0 and later.

See Also
- `setUsesSignificantDigits:` (page 934)
- `maximumSignificantDigits` (page 902)
- `minimumSignificantDigits` (page 904)

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 934).

Availability
Available in iOS 2.0 and later.

See Also
- `setZeroSymbol:` (page 934)

Declared In
`NSNumberFormatter.h`

Constants

NSNumberFormatterStyle

These constants specify predefined number format styles. These constants are used by the `numberStyle` (page 907) and `setNumberStyle:` (page 925) methods.

```
enum {
    NSNumberFormatterNoStyle = kCFNumberFormatterNoStyle,
    NSNumberFormatterDecimalStyle = kCFNumberFormatterDecimalStyle,
    NSNumberFormatterCurrencyStyle = kCFNumberFormatterCurrencyStyle,
    NSNumberFormatterPercentStyle = kCFNumberFormatterPercentStyle,
    NSNumberFormatterScientificStyle = kCFNumberFormatterScientificStyle,
    NSNumberFormatterSpellOutStyle = kCFNumberFormatterSpellOutStyle
};
typedef NSUInteger NSNumberFormatterStyle;
```

Constants

`NSNumberFormatterNoStyle`
Specifies no style.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterDecimalStyle`
Specifies a decimal style format.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterCurrencyStyle`
Specifies a currency style format.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPercentStyle`
Specifies a percent style format.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterScientificStyle`
Specifies a scientific style format.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterSpellOutStyle`
Specifies a spell-out format; for example, “23” becomes “twenty-three”.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

NSNumberFormatterBehavior

These constants specify the behavior of a number formatter. These constants are returned by the `defaultFormatterBehavior` (page 893) class method and the `formatterBehavior` (page 897) instance methods; you set them with the `setDefaultFormatterBehavior:` (page 894) class method and the `setFormatterBehavior:` (page 915) instance method.

```
enum {
    NSNumberFormatterBehaviorDefault = 0,
    NSNumberFormatterBehavior10_0 = 1000,
    NSNumberFormatterBehavior10_4 = 1040,
};
typedef NSUInteger 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 894).

Available in iOS 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 iOS 2.0 through iOS 2.1.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterBehavior10_4`

The number-formatter behavior since Mac OS X v10.4.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

NSNumberFormatterPadPosition

These constants are used to specify how numbers should be padded. These constants are used by the `paddingPosition` (page 908) and `setPaddingPosition:` (page 926) methods.

```
enum {
    NSNumberFormatterPadBeforePrefix = kCFNumberFormatterPadBeforePrefix,
    NSNumberFormatterPadAfterPrefix = kCFNumberFormatterPadAfterPrefix,
    NSNumberFormatterPadBeforeSuffix = kCFNumberFormatterPadBeforeSuffix,
    NSNumberFormatterPadAfterSuffix = kCFNumberFormatterPadAfterSuffix
};
typedef NSUInteger NSNumberFormatterPadPosition;
```

Constants

`NSNumberFormatterPadBeforePrefix`

Specifies that the padding should occur before the prefix.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadAfterPrefix`
Specifies that the padding should occur after the prefix.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadBeforeSuffix`
Specifies that the padding should occur before the suffix.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterPadAfterSuffix`
Specifies that the padding should occur after the suffix.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

NSNumberFormatterRoundingMode

These constants are used to specify how numbers should be rounded. These constants are used by the `roundingMode` (page 911)and `setRoundingMode:` (page 929) methods.

```
enum {
    NSNumberFormatterRoundCeiling = kCFNumberFormatterRoundCeiling,
    NSNumberFormatterRoundFloor = kCFNumberFormatterRoundFloor,
    NSNumberFormatterRoundDown = kCFNumberFormatterRoundDown,
    NSNumberFormatterRoundUp = kCFNumberFormatterRoundUp,
    NSNumberFormatterRoundHalfEven = kCFNumberFormatterRoundHalfEven,
    NSNumberFormatterRoundHalfDown = kCFNumberFormatterRoundHalfDown,
    NSNumberFormatterRoundHalfUp = kCFNumberFormatterRoundHalfUp
};
typedef NSUInteger NSNumberFormatterRoundingMode;
```

Constants

`NSNumberFormatterRoundCeiling`
Round up to next larger number with the proper number of digits after the decimal separator.

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundHalfDown`

Round down when a 5 follows putative last digit.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

`NSNumberFormatterRoundHalfUp`

Round up when a 5 follows putative last digit.

Available in iOS 2.0 and later.

Declared in `NSNumberFormatter.h`.

NSObject Class Reference

Inherits from	none (NSObject is a root class)
Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 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
Related sample code	BonjourWeb CryptoExercise ScrollViewSuite SpeakHere WiTap

Overview

`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 invoking a particular method. You can also ask for a method implementation and invoke it using one of the `perform...` methods, or as a function, although this is typically discouraged since it circumvents dynamic binding.

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 956) class method and the `methodForSelector:` (page 973) 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 973) 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 973) 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 973) 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 Messaging in *Objective-C Runtime Programming Guide* for more information.

Adopted Protocols

NSObject
- - autorelease (page 1629)
- - class (page 1630)
- - conformsToProtocol: (page 1630)
- - description (page 1631)
- - hash (page 1631)
- - isEqual: (page 1632)
- - isKindOfClass: (page 1632)
- - isMemberOfClass: (page 1633)
- - isProxy (page 1634)
- - performSelector: (page 1634)
- - performSelector:withObject: (page 1635)
- - performSelector:withObject:withObject: (page 1635)
- - release (page 1636)
- - respondsToSelector: (page 1637)
- - retain (page 1638)
- - retainCount (page 1638)
- - self (page 1639)
- - superclass (page 1640)
- - zone (page 1640)

Tasks

Initializing a Class

+ initialize (page 955)
Initializes the receiver before it's used (before it receives its first message).

+ load (page 958)
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 959)
Allocates a new instance of the receiving class, sends it an init (page 971) message, and returns the initialized object.

+ alloc (page 949)
Returns a new instance of the receiving class.

+ `allocWithZone:` (page 950)
  Returns a new instance of the receiving class where memory for the new instance is allocated from a given zone.

- `init` (page 971)
  Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

- `copy` (page 965)
  Returns the object returned by `copyWithZone:` (page 1554), where the zone is `nil`.

+ `copyWithZone:` (page 954)
  Returns the receiver.

- `mutableCopy` (page 974)
  Returns the object returned by `mutableCopyWithZone:` (page 1614) where the zone is `nil`.

+ `mutableCopyWithZone:` (page 959)
  Returns the receiver.

- `dealloc` (page 966)
  Deallocates the memory occupied by the receiver.

- `finalize` (page 968)
  The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Identifying Classes

+ `class` (page 952)
  Returns the class object.

+ `superclass` (page 962)
  Returns the class object for the receiver's superclass.

+ `isSubclassOfClass:` (page 958)
  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 957)
  Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

Testing Protocol Conformance

+ `conformsToProtocol:` (page 953)
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Obtaining Information About Methods

- `methodForSelector:` (page 973)
  Locates and returns the address of the receiver's implementation of a method so it can be called as a function.
+ `instanceMethodForSelector:` (page 956)
  Locates and returns the address of the implementation of the instance method identified by a given selector.
+ `instanceMethodSignatureForSelector:` (page 957)
  Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.
- `methodSignatureForSelector:` (page 974)
  Returns an `NSMethodSignature` object that contains a description of the method identified by a given selector.

Describing Objects

+ `description` (page 954)
  Returns a string that represents the contents of the receiving class.

Discardable Content Proxy Support

- `autoContentAccessingProxy` (page 963)
  Creates and returns an autoreleased proxy for the receiving object

Sending Messages

- `performSelector:withObject:afterDelay:` (page 977)
  Invokes a method of the receiver on the current thread using the default mode after a delay.
- `performSelector:withObject:afterDelay:inModes:` (page 978)
  Invokes a method of the receiver on the current thread using the specified modes after a delay.
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 980)
  Invokes a method of the receiver on the main thread using the default mode.
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 981)
  Invokes a method of the receiver on the main thread using the specified modes.
- `performSelector:onThread:withObject:waitUntilDone:` (page 975)
  Invokes a method of the receiver on the specified thread using the default mode.
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 976)
  Invokes a method of the receiver on the specified thread using the specified modes.
- `performSelectorInBackground:withObject:` (page 979)
  Invokes a method of the receiver on a new background thread.
+ `cancelPreviousPerformRequestsWithTarget:` (page 950)
  Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) instance method.

+ cancelPreviousPerformRequestsWithTarget:selector:object: (page 951)
Cancels perform requests previously registered with performSelector:withObject:afterDelay: (page 977).

Forwarding Messages

- forwardingTargetForSelector: (page 969)
Returns the object to which unrecognized messages should first be directed.
- forwardInvocation: (page 970)
Overridden by subclasses to forward messages to other objects.

Dynamically Resolving Methods

+ resolveClassMethod: (page 960)
Dynamically provides an implementation for a given selector for a class method.
+ resolveInstanceMethod: (page 961)
Dynamically provides an implementation for a given selector for an instance method.

Error Handling

- doesNotRecognizeSelector: (page 967)
Handles messages the receiver doesn't recognize.

Archiving

- awakeAfterUsingCoder: (page 964)
Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.
- classForCoder (page 964)
Overridden by subclasses to substitute a class other than its own during coding.
- classForKeyedArchiver (page 965)
Overridden by subclasses to substitute a new class for instances during keyed archiving.
+ classFallbacksForKeyedArchiver (page 952)
Overridden to return the names of classes that can be used to decode objects if their class is unavailable.
+ classForKeyedUnarchiver (page 953)
Overridden by subclasses to substitute a new class during keyed unarchiving.
- replacementObjectForCoder: (page 982)
Overridden by subclasses to substitute another object for itself during encoding.
- replacementObjectForKeyedArchiver: (page 982)
Overridden by subclasses to substitute another object for itself during keyed archiving.
+ setVersion: (page 962)
Sets the receiver's version number.

`+ version` (page 962)
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 950) 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 1636) or `autorelease` (page 1629).

Availability
Available in iOS 2.0 and later.

See Also
`- init` (page 971)

Related Sample Code
BonjourWeb
CryptoExercise
ScrollViewSuite
SpeakHere
WiTap

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 1640) 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 1636) or `autorelease` (page 1629).

Availability

Available in iOS 2.0 and later.

See Also

`+ alloc` (page 949)

`- init` (page 971)

Declared In

`NSObject.h`

cancelPreviousPerformRequestsWithTarget:

Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) instance method.

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) 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 iOS 2.0 and later.

Declared In

`NSRunLoop.h`

cancelPreviousPerformRequestsWithTarget:selector:object:

Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 977).

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget selector:(SEL)aSelector
    object:(id)anArgument
```

Parameters

aTarget

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) instance method

aSelector

The selector for requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) instance method.

See "Selectors" (page 943) for a description of the `SEL` type.

anArgument

The argument for requests previously registered with the `performSelector:withObject:afterDelay:` (page 977) instance method. Argument equality is determined using `isEqual:` (page 1632), 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 iOS 2.0 and later.

Related Sample Code

ScrollViewSuite

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 1632) method (declared in the `NSObject` protocol):

```
BOOL test = [self isKindOfClass:[SomeClass class]];
```

Availability
Available in iOS 2.0 and later.

See Also
`class` (page 1630) (`NSObject` protocol)

Related Sample Code
aurioTouch
GLSprite
SpeakHere

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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
`+ conformsToProtocol:` (page 953)

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 iOS 2.0 and later.

See Also
`- copy` (page 965)

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 iOS 2.0 and later.

See Also
`description` (page 1631) (`NSObject` protocol)

Related Sample Code
SpeakHere

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 959) 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 957) 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.

Special Considerations

`initialize` it is invoked only once per class. If you want to perform independent initialization for the class and for categories of the class, you should implement `load` (page 958) methods.

Availability

Available in iOS 2.0 and later.

See Also

`- init` (page 971)

`+ load` (page 958)

`class` (page 1630) (`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 1637) method to check before passing the selector to `methodForSelector:`.

See "Selectors" (page 943) 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 973) instance method to the class instead.

See "Selectors" (page 943) for a description of the `IMP` type, and how to invoke the returned method implementation.

Availability
Available in iOS 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 943) 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 iOS 2.0 and later.

See Also
- methodSignatureForSelector: (page 974)

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 943) 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 1637).

Availability
Available in iOS 2.0 and later.

See Also
- forwardInvocation: (page 970)

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 iOS 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 iOS 2.0 and later.

See Also
`+ initialize` (page 955)

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 iOS 2.0 and later.

Declared In
`NSObject.h`

new

Allocates a new instance of the receiving class, sends it an `init` (page 971) 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 949) and `init` (page 971). Like `alloc` (page 949), it initializes the `isa` instance variable of the new object so it points to the class data structure. It then invokes the `init` (page 971) method to complete the initialization process.

Unlike `alloc` (page 949), `new` (page 959) 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 949) 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 retain];
    return [[self alloc] initWithTag:tag data:data];
}
```

Although it's appropriate to define new `new...` methods in this way, the `alloc` (page 949) and `allocWithZone:` (page 950) 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 is *not* autoreleased. The invoker of this method is responsible for releasing the returned object, using either `release` (page 1636) or `autorelease` (page 1629).

Availability

Available in iOS 2.0 and later.

Related Sample Code

WiTap

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 provide an implementation for a given selector. See `resolveInstanceMethod:` (page 961) for further discussion.

Availability

Available in iOS 2.0 and later.

See Also
`+ resolveInstanceMethod:` (page 961)

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 960) 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 Message Forwarding in *Objective-C Runtime Programming Guide*) is invoked. If `respondsToSelector:` (page 1637) or `instancesRespondToSelector:` (page 957) is invoked, the dynamic method resolver is given the opportunity to provide an `IMP` for the given selector first.

Availability
Available in iOS 2.0 and later.

See Also
`+ resolveClassMethod:` (page 960)

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 iOS 2.0 and later.

See Also
+ version (page 962)

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 iOS 2.0 and later.

See Also
+ class (page 952)
superclass (page 1640) (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 iOS 2.0 and later.

See Also
`+ setVersion:` (page 962)
`versionForClassName:` (page 219) (`NSCoder`)

Declared In
`NSObject.h`

Instance Methods

autoContentAccessingProxy

Creates and returns an autoreleased proxy for the receiving object

```
- (id)autoContentAccessingProxy
```

Return Value
An autoreleased proxy of the receiver.

Discussion
This method creates and returns an autoreleased proxy for the receiving object, if the receiver adopts the `NSDiscardableContent` protocol and still has undiscarded content.

The proxy calls `beginContentAccess` (page 1562) on the receiver to keep the content available as long as the proxy lives, and calls `endContentAccess` (page 1563) when the proxy is deallocated (or finalized).

The wrapper object is otherwise a subclass of `NSProxy` and forwards messages to the original receiver object as an NSProxy does.

This method can be used to hide an `NSDiscardableContent` object's content volatility by creating an object that responds to the same messages but holds the contents of the original receiver available as long as the created proxy lives. Thus hidden, the `NSDiscardableContent` object (by way of the proxy) can be given out to unsuspecting recipients of the object who would otherwise not know they might have to call `beginContentAccess` (page 1562) and `endContentAccess` (page 1563) around particular usages (specific to each `NSDiscardableContent` object) of the `NSDiscardableContent` object.

Availability
Available in iOS 4.0 and later.

Declared In
`NSObject.h`

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 iOS 2.0 and later.

See Also
`- classForCoder` (page 964)
`- replacementObjectForCoder:` (page 982)
`initWithCoder:` (page 1552) (`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 iOS 2.0 and later.

See Also
- `awakeAfterUsingCoder:` (page 964)
- `replacementObjectForCoder:` (page 982)

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 iOS 2.0 and later.

See Also
- `replacementObjectForKeyedArchiver:` (page 982)

Declared In
`NSKeyedArchiver.h`

copy

Returns the object returned by `copyWithZone:` (page 1554), where the zone is `nil`.

```
- (id)copy
```

Return Value
The object returned by the `NSCopying` protocol method `copyWithZone:` (page 1554), 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 1554).

`NSObject` does not itself support the `NSCopying` protocol. Subclasses must support the protocol and implement the `copyWithZone:` (page 1554) method. A subclass version of the `copyWithZone:` (page 1554) 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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
GKRocket
WiTap

Declared In
`NSObject.h`

dealloc

Deallocates the memory occupied by the receiver.

```
- (void)dealloc
```

Discussion
Subsequent messages to the receiver may 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 1636) `NSObject` protocol method (if the `release` message results in the receiver's retain count becoming `0`). See *Memory Management Programming Guide* 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];
}
```

Important: 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. For this and other reasons, you should not manage scarce resources in `dealloc`—see Object Ownership and Disposal in *Memory Management Programming Guide* for more details.

Special Considerations

When garbage collection is enabled, the garbage collector sends `finalize` (page 968) to the receiver instead of `dealloc`.

When garbage collection is enabled, this method is a no-op.

Availability

Available in iOS 2.0 and later.

See Also

`autorelease` (page 1629) (`NSObject` protocol)

`release` (page 1636) (`NSObject` protocol)

`- finalize` (page 968)

Related Sample Code

BonjourWeb

CryptoExercise

GKRocket

ScrollViewSuite

SpeakHere

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 943) 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 965) or `init` (page 971) method by re-implementing it to include a `doesNotRecognizeSelector:` message as follows:

```
- (id)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 1637) will still report that the receiver has access to a `copy` method.

If you override this method, you must call `super` or raise an `NSInvalidArgumentException` (page 1773) exception at the end of your implementation. In other words, this method must not return normally; it must always result in an exception being thrown.

Availability
Available in iOS 2.0 and later.

See Also
`- forwardInvocation:` (page 970)

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 iOS 2.0 and later.

See Also

`- dealloc` (page 966)

Declared In

`NSObject.h`

forwardingTargetForSelector:

Returns the object to which unrecognized messages should first be directed.

```
- (id)forwardingTargetForSelector:(SEL)aSelector
```

Parameters

aSelector

A selector for a method that the receiver does not implement.

Return Value

The object to which unrecognized messages should first be directed.

Discussion

If an object implements (or inherits) this method, and returns a non-`nil` (and non-`self`) result, that returned object is used as the new receiver object and the message dispatch resumes to that new object. (Obviously if you return `self` from this method, the code would just fall into an infinite loop.)

If you implement this method in a non-root class, if your class has nothing to return for the given selector then you should return the result of invoking super's implementation.

This method gives an object a chance to redirect an unknown message sent to it before the much more expensive `forwardInvocation:` (page 970) machinery takes over. This is useful in basic proxying situations and can be an order of magnitude faster than regular forwarding. It is not useful where the goal of the forwarding is to capture the NSInvocation, or manipulate the arguments or return value during the forwarding.

Availability

Available in iOS 4.0 and later.

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 974) in addition to `forwardInvocation:`. The mechanism for forwarding messages uses information obtained from `methodSignatureForSelector:` (page 974) 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 967) 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 iOS 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 949) or `allocWithZone:` (page 950) 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 949) or `allocWithZone:` (page 950), 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`:

```
- (id)init
{
    self = [super init];
    if (self) {
        /* 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:

```
- (id)init;
- (id)initWithTag:(int)tag;
- (id)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:

```
- (id)init
{
    return [self initWithTag:-1];
}

- (id)initWithTag:(int)tag
{
    return [self initWithTag:tag data:NULL];
}

- (id)initWithTag:(int)tag data:(struct info *)data
{
    self = [super init. . .];
    if (self) {
        /* 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:

```
- (id)initWithTag:(int)tag data:(struct info *)data object:anObject
{
    self = [super initWithTag:tag data:data];
    if (self) {
        /* 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:

```
- (id)initWithTag:(int)tag data:(struct info *)data
{
    return [self initWithTag:tag data:data object: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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise
FastEnumerationSample
GKRocket
WiTap

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 1637) 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 943) for a description of the `IMP` and `SEL` types, and how to invoke the returned method implementation.

Availability
Available in iOS 2.0 and later.

See Also
`+ instanceMethodForSelector:` (page 956)

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 943) 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 iOS 2.0 and later.

See Also
`+ instanceMethodSignatureForSelector:` (page 957)
`- forwardInvocation:` (page 970)

Declared In
`NSObject.h`

mutableCopy

Returns the object returned by `mutableCopyWithZone:` (page 1614) where the zone is `nil`.

```
- (id)mutableCopy
```

Return Value
The object returned by the `NSMutableCopying` protocol method `mutableCopyWithZone:` (page 1614), 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 1614).

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 iOS 2.0 and later.

Related Sample Code
KeyboardAccessory
ToolbarSearch

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 943) 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` (page 1113) 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 977) or `performSelector:withObject:afterDelay:inModes:` (page 978) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 976)
- `performSelectorInBackground:withObject:` (page 979)

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 943) 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 977) or `performSelector:withObject:afterDelay:inModes:` (page 978) method instead.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
- `performSelector:onThread:withObject:waitUntilDone:` (page 975)
- `performSelectorInBackground:withObject:` (page 979)

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 943) 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. Specifying a delay of 0 does not necessarily cause the selector to be performed immediately. The selector is still queued on the thread's run loop and performed as soon as possible.

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 978) method instead. To ensure that the selector is performed on the main thread, use the

performSelectorOnMainThread:withObject:waitUntilDone: (page 980) or performSelectorOnMainThread:withObject:waitUntilDone:modes: (page 981) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 950) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 951) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
+ cancelPreviousPerformRequestsWithTarget:selector:object: (page 951)
- performSelectorOnMainThread:withObject:waitUntilDone: (page 980)
- performSelectorOnMainThread:withObject:waitUntilDone:modes: (page 981)
- performSelector:onThread:withObject:waitUntilDone:modes: (page 976)

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 943) 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. Specifying a delay of 0 does not necessarily cause the selector to be performed immediately. The selector is still queued on the thread's run loop and performed as soon as possible.

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 978) method instead. To ensure that the selector is performed on the main thread, use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 980) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 981) method instead. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 950) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 951) method.

This method retains the receiver and the *anArgument* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
- `performSelector:withObject:afterDelay:` (page 977)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 980)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 981)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 976)

`addTimer:forMode:` (page 1107) (`NSRunLoop`)
`invalidate` (page 1332) (`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 943) 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 iOS 2.0 and later.

See Also
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 976)

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 943) 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 (page 1113) 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 977) or performSelector:withObject:afterDelay:inModes: (page 978) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
- performSelector:withObject:afterDelay: (page 977)
- performSelector:withObject:afterDelay:inModes: (page 978)
- performSelectorOnMainThread:withObject:waitUntilDone:modes: (page 981)
- performSelector:onThread:withObject:waitUntilDone:modes: (page 976)

Related Sample Code
CryptoExercise

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 943) 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 977) or performSelector:withObject:afterDelay:inModes: (page 978) method.

This method retains the receiver and the *arg* parameter until after the selector is performed.

Availability
Available in iOS 2.0 and later.

See Also
- performSelector:withObject:afterDelay: (page 977)
- performSelector:withObject:afterDelay:inModes: (page 978)
- performSelectorOnMainThread:withObject:waitUntilDone: (page 980)

- performSelector:onThread:withObject:waitUntilDone:modes: (page 976)

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 iOS 2.0 and later.

See Also
- classForCoder (page 964)
- awakeAfterUsingCoder: (page 964)

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 iOS 2.0 and later.

See Also
- classForKeyedArchiver (page 965)

Declared In
NSKeyedArchiver.h

NSOperation Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Concurrency Programming Guide

Overview

The `NSOperation` class is an abstract class you use to encapsulate the code and data associated with a single task. Because it is abstract, you do not use this class directly but instead subclass or use one of the system-defined subclasses (`NSInvocationOperation` or `NSBlockOperation`) to perform the actual task. Despite being abstract, the base implementation of `NSOperation` does include significant logic to coordinate the safe execution of your task. The presence of this built-in logic allows you to focus on the actual implementation of your task, rather than on the glue code needed to ensure it works correctly with other system objects.

An operation object is a single-shot object—that is, it executes its task once and cannot be used to execute it again. You typically execute operations by adding them to an operation queue (an instance of the `NSOperationQueue` class). An operation queue executes its operations either directly, by running them on secondary threads, or indirectly using the `libdispatch` library (also known as Grand Central Dispatch). For more information about how queues execute operations, see *NSOperationQueue Class Reference*.

If you do not want to use an operation queue, you can execute an operation yourself by calling its `start` method directly from your code. Executing operations manually does put more of a burden on your code, because starting an operation that is not in the ready state triggers an exception. The `isReady` method reports on the operation's readiness.

Operation Dependencies

Dependencies are a convenient way to execute operations in a specific order. You can add and remove dependencies for an operation using the `addDependency:` and `removeDependency:` methods. By default, an operation object that has dependencies is not considered ready until all of its dependent operation objects have finished executing. Once the last dependent operation finishes, however, the operation object becomes ready and able to execute.

The dependencies supported by `NSOperation` make no distinction about whether a dependent operation finished successfully or unsuccessfully. (In other words, canceling an operation similarly marks it as finished.) It is up to you to determine whether an operation with dependencies should proceed in cases where its dependent operations were cancelled or did not complete their task successfully. This may require you to incorporate some additional error tracking capabilities into your operation objects.

KVO-Compliant Properties

The `NSOperation` class is key-value coding (KVC) and key-value observing (KVO) compliant for several of its properties. As needed, you can observe these properties to control other parts of your application. The properties you can observe 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
- `completionBlock` - readable and writable property (Mac OS X only)

Although you can attach observers to these properties, you should not use Cocoa bindings to bind them to elements of your application's user interface. Code associated with your user interface typically must execute only in your application's main thread. Because an operation may execute in any thread, KVO notifications associated with that operation may similarly occur in any thread.

If you provide custom implementations for 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*.

Multicore Considerations

The `NSOperation` class is itself multicore aware. It is therefore safe to call the methods of an `NSOperation` object from multiple threads without creating additional locks to synchronize access to the object. This behavior is necessary because an operation typically runs in a separate thread from the one that created and is monitoring it.

When you subclass `NSOperation`, you must make sure that any overridden methods remain safe to call from multiple threads. If you implement custom methods in your subclass, such as custom data accessors, you must also make sure those methods are thread-safe. Thus, access to any data variables in the operation must be synchronized to prevent potential data corruption. For more information about synchronization, see *Threading Programming Guide*.

Concurrent Versus Non-Concurrent Operations

If you plan on executing an operation object manually, instead of adding it to a queue, you can design your operation to execute in a concurrent or non-concurrent manner. Operation objects are non-concurrent by default. In a non-concurrent operation, the operation's task is performed synchronously—that is, the operation object does not create a separate thread on which to run the task. Thus, when you call the `start` method of a non-concurrent operation directly from your code, the operation executes immediately in the current thread. By the time the `start` method of such an object returns control to the caller, the task itself is complete.

In contrast to a non-concurrent operation, which runs synchronously, a concurrent operation runs asynchronously. In other words, when you call the `start` method of a concurrent operation, that method could return before the corresponding task is completed. This might happen because the operation object created a new thread to execute the task or because the operation called an asynchronous function. It does not actually matter if the operation is ongoing when control returns to the caller, only that it could be ongoing.

If you always plan to use queues to execute your operations, it is simpler to define them as non-concurrent. If you execute operations manually, though, you might want to define your operation objects as concurrent to ensure that they always execute asynchronously. Defining a concurrent operation requires more work, because you have to monitor the ongoing state of your task and report changes in that state using KVO notifications. But defining concurrent operations can be useful in cases where you want to ensure that a manually executed operation does not block the calling thread.

For information on how to define both concurrent and non-concurrent operations, see the subclassing notes.

Note: In Mac OS X v10.6, operation queues ignore the value returned by `isConcurrent` and always call the `start` method of your operation from a separate thread. In Mac OS X v10.5, however, operation queues create a thread only if `isConcurrent` returns `NO`. In general, if you are always using operations with an operation queue, there is no reason to make them concurrent.

Subclassing Notes

The `NSOperation` class provides the basic logic to track the execution state of your operation but otherwise must be subclassed to do any real work. How you create your subclass depends on whether your operation is designed to execute concurrently or non-concurrently.

Methods to Override

For non-concurrent operations, you typically override only one method:

- `main`

Into this method, you place the code needed to perform the given task. Of course, you should also define a custom initialization method to make it easier to create instances of your custom class. You might also want to define getter and setter methods to access the data from the operation. However, if you do define custom getter and setter methods, you must make sure those methods can be called safely from multiple threads.

If you are creating a concurrent operation, you need to override the following methods at a minimum:

- `start`

- `isConcurrent`
- `isExecuting`
- `isFinished`

In a concurrent operation, your `start` method is responsible for starting the operation in an asynchronous manner. Whether you spawn a thread or call an asynchronous function, you do it from this method. Upon starting the operation, your `start` method should also update the execution state of the operation as reported by the `isExecuting` method. You do this by sending out KVO notifications for the `isExecuting` key path, which lets interested clients know that the operation is now running. Your `isExecuting` method must also return the status in a thread-safe manner.

Upon completion or cancellation of its task, your concurrent operation object must generate KVO notifications for both the `isExecuting` and `isFinished` key paths to mark the final change of state for your operation. (In the case of cancellation, it is still important to update the `isFinished` key path, even if the operation did not completely finish its task. Queued operations must report that they are finished before they can be removed from a queue.) In addition to generating KVO notifications, your overrides of the `isExecuting` and `isFinished` methods should also continue to return accurate values based on the state of your operation.

For additional information and guidance on how to define concurrent operations, see *Concurrency Programming Guide*.

Important: At no time in your `start` method should you ever call `super`. When you define a concurrent operation, you take it upon yourself to provide the same behavior that the default `start` method provides, which includes starting the task and generating the appropriate KVO notifications. Your `start` method should also check to see if the operation itself was cancelled before actually starting the task. For more information about cancellation semantics, see "Responding to the Cancel Command" (page 989).

Even for concurrent operations, there should be little need to override methods other than those described above. However, if you customize the dependency features of operations, you might have to override additional methods and provide additional KVO notifications. In the case of dependencies, this would likely only require providing notifications for the `isReady` key path. Because the `dependencies` property is used to manage the list of dependent operations, changes to it are already handled by the default `NSOperation` class.

Maintaining Operation Object States

Operation objects maintain state information internally to determine when it is safe to execute and also to notify external clients of the progression through the operation's life cycle. Your custom subclasses must maintain this state information to ensure the correct execution of operations in your code. Table 67-1 lists the key paths associated with an operation's states and how you should manage that key path in any custom subclasses.

Table 67-1 Key paths for operation object states

Key Path	Description
`isReady`	The `isReady` key path lets clients know when an operation is ready to execute. The `isReady` method returns `YES` to indicate that the operation is ready to execute now or `NO` if there are still unfinished operations on which it is dependent. In most cases, you do not have to manage the state of this key path yourself. If the readiness of your operations is determined by factors other than dependent operations, however—such as by some external condition in your program—you can provide your own implementation of the `isReady` method and track your operation's readiness yourself. It is often simpler though just to create operation objects only when your external state allows it. In Mac OS X v10.6 and later, if you cancel an operation while it is waiting on the completion of one or more dependent operations, those dependencies are thereafter ignored and the value of this property is updated to reflect that it is now ready to run. This behavior gives an operation queue the chance to flush cancelled operations out of its queue more quickly.
`isExecuting`	The `isExecuting` key path lets clients know whether the operation is actively working on its assigned task. The `isExecuting` method must return `YES` if it is working on its task or `NO` if it is not. If you replace the `start` method of your operation object, you must also replace the `isExecuting` method and generate KVO notifications when the execution state of your operation changes.
`isFinished`	The `isFinished` key path lets clients know that an operation finished its task successfully or was cancelled and is exiting. An operation object does not clear a dependency until the value at the `isFinished` key path changes to `YES`. Similarly, an operation queue does not dequeue an operation until the `isFinished` method returns `YES`. Thus, marking operations as finished is critical to keeping queues from backing up with in-progress or cancelled operations. If you replace the `start` method or your operation object, you must also replace the `isFinished` method and generate KVO notifications when the operation finishes executing or is cancelled.
`isCancelled`	The `isCancelled` key path lets clients know that the cancellation of an operation was requested. Support for cancellation is voluntary but encouraged and your own code should not have to send KVO notifications for this key path. The handling of cancellation notices in an operation is described in more detail in "Responding to the Cancel Command" (page 989).

Responding to the Cancel Command

Once you add an operation to a queue, the operation is out of your hands. The queue takes over and handles the scheduling of that task. However, if you decide later that you do not want to execute the operation after all—because the user pressed a cancel button in a progress panel or quit the application, for example—you can cancel the operation to prevent it from consuming CPU time needlessly. You do this by calling the `cancel` method of the operation object itself or by calling the `cancelAllOperations` (page 1008) method of the `NSOperationQueue` class.

Cancelling an operation does not immediately force it to stop what it is doing. Although respecting the value returned by the `isCancelled` is expected of all operations, your code must explicitly check the value returned by this method and abort as needed. The default implementation of `NSOperation` does include checks for cancellation. For example, if you cancel an operation before its `start` method is called, the `start` method exits without starting the task.

Note: In Mac OS X v10.6, the behavior of the `cancel` method varies depending on whether the operation is currently in an operation queue. For unqueued operations, this method marks the operation as finished immediately, generating the appropriate KVO notifications. For queued operations, it simply marks the operation as ready to execute and lets the queue call its `start` method, which subsequently exits and results in the clearing of the operation from the queue.

You should always support cancellation semantics in any custom code you write. In particular, your main task code should periodically check the value of the `isCancelled` method. If the method ever returns `YES`, your operation object should clean up and exit as quickly as possible. If you implement a custom `start` method, that method should include early checks for cancellation and behave appropriately. Your custom `start` method must be prepared to handle this type of early cancellation.

In addition to simply exiting when an operation is cancelled, it is also important that you move a cancelled operation to the appropriate final state. Specifically, if you manage the values for the `isFinished` and `isExecuting` properties yourself (perhaps because you are implementing a concurrent operation), you must update those variables accordingly. Specifically, you must change the value returned by `isFinished` to `YES` and the value returned by `isExecuting` to `NO`. You must make these changes even if the operation was cancelled before it started executing.

Tasks

Initialization

- `init` (page 994)
  Returns an initialized `NSOperation` object.

Executing the Operation

- `start` (page 1000)
  Begins the execution of the operation.
- `main` (page 997)
  Performs the receiver's non-concurrent task.
- `completionBlock` (page 993)
  Returns the block to execute when the operation's main task is complete.
- `setCompletionBlock:` (page 998)
  Sets the block to execute when the operation has finished executing.

Canceling Operations

- `cancel` (page 992)
  Advises the operation object that it should stop executing its task.

Getting the Operation Status

- `isCancelled` (page 994)
  Returns a Boolean value indicating whether the operation has been cancelled.
- `isExecuting` (page 995)
  Returns a Boolean value indicating whether the operation is currently executing.
- `isFinished` (page 996)
  Returns a Boolean value indicating whether the operation is done executing.
- `isConcurrent` (page 995)
  Returns a Boolean value indicating whether the operation runs asynchronously.
- `isReady` (page 996)
  Returns a Boolean value indicating whether the receiver's operation can be performed now.

Managing Dependencies

- `addDependency:` (page 992)
  Makes the receiver dependent on the completion of the specified operation.
- `removeDependency:` (page 997)
  Removes the receiver's dependence on the specified operation.
- `dependencies` (page 993)
  Returns a new array object containing the operations on which the receiver is dependent.

Prioritizing Operations in an Operation Queue

- `queuePriority` (page 997)
  Returns the priority of the operation in an operation queue.
- `setQueuePriority:` (page 998)
  Sets the priority of the operation when used in an operation queue.

Managing the Execution Priority

- `threadPriority` (page 1000)
  Returns the thread priority to use when executing the operation.
- `setThreadPriority:` (page 999)
  Sets the thread priority to use when executing the operation.

Waiting for Completion

`- waitUntilFinished` (page 1001)
Blocks execution of the current thread until the receiver finishes.

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 should depend. 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 have finished executing. If the receiver is already executing its task, adding dependencies has no 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 iOS 2.0 and later.

See Also
`- removeDependency:` (page 997)
`- dependencies` (page 993)

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. Instead, it updates the object's internal flags to reflect the change in state. If the operation has already finished executing, this method has no effect. Canceling an operation that is currently in an operation queue, but not yet executing, makes it possible to remove the operation from the queue sooner than usual.

In Mac OS X v10.6 and later, if an operation is in a queue but waiting on unfinished dependent operations, those operations are subsequently ignored. Because it is already cancelled, this behavior allows the operation queue to call the operation's `start` method sooner and clear the object out of the queue. If you cancel an operation that is not in a queue, this method immediately marks the object as finished. In each case, marking the object as ready or finished results in the generation of the appropriate KVO notifications.

In versions of Mac OS X prior to 10.6, an operation object remains in the queue until all of its dependencies are removed through the normal processes. Thus, the operation must wait until all of its dependent operations finish executing or are themselves cancelled and have their `start` method called.

For more information on what you must do in your operation objects to support cancellation, see "Responding to the Cancel Command" (page 989).

Availability
Available in iOS 2.0 and later.

See Also
- `isCancelled` (page 994)

Declared In
`NSOperation.h`

completionBlock

Returns the block to execute when the operation's main task is complete.

```
- (void (^)(void))completionBlock
```

Return Value
The block to execute after the operation's main task is completed. This block takes no parameters and has no return value.

Discussion
Operation objects monitor the `isFinished` key path and execute this block when the value at that key path changes to `YES`. As a result, this block is called regardless of whether the operation completed successfully or was cancelled.

Availability
Available in iOS 4.0 and later.

See Also
- `setCompletionBlock:` (page 998)

Declared In
`NSOperation.h`

dependencies

Returns a new array object containing the operations on which the receiver is dependent.

```
- (NSArray *)dependencies
```

Return Value
A new array object containing the `NSOperation` objects.

Discussion
The receiver is not considered ready to execute until all of its dependent operations finish executing.

Operations are not removed from this dependency list as they finish executing. You can therefore use this list to track all dependent operations, including those that have already finished executing. The only way to remove an operation from this list is to use the `removeDependency:` method.

Availability
Available in iOS 2.0 and later.

See Also
- `addDependency:` (page 992)
- `removeDependency:` (page 997)

Declared In
`NSOperation.h`

init

Returns an initialized `NSOperation` object.

```
- (id)init
```

Return Value
The initialized `NSOperation` object.

Discussion
Your custom subclasses must call this method. The default implementation initializes the object's instance variables and prepares the it for use.

Availability
Available in iOS 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`.

You should always call this method before doing any work towards accomplishing the operation's task, which typically means calling it at the beginning of your custom `main` method. It is possible for an operation to be cancelled before it begins executing or at any time while it is executing. Therefore, calling this method at the beginning of your `main` method (and periodically throughout that method) lets you exit as quickly as possible when an operation is cancelled.

Availability
Available in iOS 2.0 and later.

See Also
`- cancel` (page 992)

Declared In
`NSOperation.h`

isConcurrent

Returns a Boolean value indicating whether the operation runs asynchronously.

```
- (BOOL)isConcurrent
```

Return Value
`YES` if the operation runs asynchronously with respect to the current thread or `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 987).

In Mac OS X v10.6 and later, operation queues ignore the value returned by this method and always start operations on a separate thread.

Availability
Available in iOS 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. If you do override it, be sure to generate KVO notifications for the `isExecuting` key path whenever the execution state of your operation object changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in iOS 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 and return a Boolean to indicate whether your operation is currently finished. If you do override it, be sure to generate appropriate KVO notifications for the `isFinished` key path when the completion state of your operation object changes. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

Availability
Available in iOS 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.

If you want to use custom conditions to determine the readiness of your operation object, you can override this method and return a value that accurately reflects the readiness of the receiver. If you do so, your custom implementation should invoke `super` and incorporate its return value into the readiness state of the object. Your custom implementation must also generate appropriate KVO notifications for the `isReady` key path.

Availability
Available in iOS 2.0 and later.

See Also
`- dependencies` (page 993)

Declared In
NSOperation.h

main

Performs the receiver's non-concurrent task.

```
- (void)main
```

Discussion
The default implementation of this method does nothing. You should override this method to perform the desired task. In your implementation, do not invoke `super`.

If you are implementing a concurrent operation, you are not required to override this method but may do so if you plan to call it from your custom `start` method.

Availability
Available in iOS 2.0 and later.

See Also
`- start` (page 1000)

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 1002).) If no priority is explicitly set, this method returns `NSOperationQueuePriorityNormal`.

Availability
Available in iOS 2.0 and later.

See Also
`- setQueuePriority:` (page 998)

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 iOS 2.0 and later.

See Also

- `addDependency:` (page 992)
- `dependencies` (page 993)

Declared In

`NSOperation.h`

setCompletionBlock:

Sets the block to execute when the operation has finished executing.

```
- (void)setCompletionBlock:(void (^)(void))block
```

Parameters

block

The block to be executed when the operation finishes. This method creates a copy of the specified block. The block itself should take no parameters and have no return value.

Discussion

The exact execution context for your completion block is not guaranteed but is typically a secondary thread. Therefore, you should not use this block to do any work that requires a very specific execution context. Instead, you should shunt that work to your application's main thread or to the specific thread that is capable of doing it. For example, if you have a custom thread for coordinating the completion of the operation, you could use the completion block to ping that thread.

A finished operation may finish either because it was cancelled or because it successfully completed its task. You should take that fact into account when writing your block code. Similarly, you should not make any assumptions about the successful completion of dependent operations, which may themselves have been cancelled.

Availability

Available in iOS 4.0 and later.

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 1002).

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 iOS 2.0 and later.

See Also

- `queuePriority` (page 997)
- `addDependency:` (page 992)

Declared In

`NSOperation.h`

setThreadPriority:

Sets the thread priority to use when executing the operation.

```
- (void)setThreadPriority:(double)priority
```

Parameters

priority

The new thread priority, specified as a floating-point number in the range 0.0 to 1.0, where 1.0 is the highest priority.

Discussion

The value you specify is mapped to the operating system's priority values. The specified thread priority is applied to the thread only while the operation's `main` method is executing. It is not applied while the operation's completion block is executing. For a concurrent operation in which you create your own thread, you must set the thread priority yourself in your custom `start` method and reset the original priority when the operation is finished.

Availability

Available in iOS 4.0 and later.

See Also

+ `setThreadPriority:` (page 1315) (NSThread)

Declared In

`NSOperation.h`

start

Begins the execution of the operation.

```
- (void)start
```

Discussion
The default implementation of this method updates the execution state of the operation and calls the receiver's main method. This method also performs several checks to ensure that the operation can actually run. For example, if the receiver was cancelled or is already finished, this method simply returns without calling main. (In Mac OS X v10.5, this method throws an exception if the operation is already finished.) If the operation is currently executing or is not ready to execute, this method throws an `NSInvalidArgumentException` exception. In Mac OS X v10.5, this method catches and ignores any exceptions thrown by your main method automatically. In Mac OS X v10.6 and later, exceptions are allowed to propagate beyond this method. You should never allow exceptions to propagate out of your main method.

Note: An operation is not considered ready to execute if it is still dependent on other operations that have not yet finished.

If you are implementing a concurrent operation, you must override this method and use it to initiate your operation. Your custom implementation must not call `super` at any time. In addition to configuring the execution environment for your task, your implementation of this method must also track the state of the operation and provide appropriate state transitions. When the operation executes and subsequently finishes its work, it should generate KVO notifications for the `isExecuting` and `isFinished` key paths respectively. For more information about manually generating KVO notifications, see *Key-Value Observing Programming Guide*.

You can call this method explicitly if you want to execute your operations manually. However, it is a programmer error to call this method on an operation object that is already in an operation queue or to queue the operation after calling this method. Once you add an operation object to a queue, the queue assumes all responsibility for it.

Availability
Available in iOS 2.0 and later.

See Also
- `main` (page 997)
- `isReady` (page 996)
- `dependencies` (page 993)

Declared In
`NSOperation.h`

threadPriority

Returns the thread priority to use when executing the operation.

```
- (double)threadPriority
```

Return Value
A floating-point number in the range 0.0 to 1.0, where 1.0 is the highest priority. The default thread priority is 0.5.

Availability
Available in iOS 4.0 and later.

See Also
`+ threadPriority` (page 1316)

Declared In
`NSOperation.h`

waitUntilFinished

Blocks execution of the current thread until the receiver finishes.

```
- (void)waitUntilFinished
```

Discussion
The receiver should never call this method on itself and should avoid calling it on any operations submitted to the same operation queue as itself. Doing so can cause the operation to deadlock. It is generally safe to call this method on an operation that is in a different operation queue, although it is still possible to create deadlocks if each operation waits on the other.

A typical use for this method would be to call it from the code that created the operation in the first place. After submitting the operation to a queue, you would call this method to wait until that operation finished executing.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

Constants

NSOperationQueuePriority

Describes the priority of an operation relative to other operations in an operation queue.

```
typedef NSInteger NSOperationQueuePriority;
```

Discussion
For a list of related constants, see "Operation Priorities" (page 1002).

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityLow`
Operations receive low priority for execution.
Available in iOS 2.0 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityNormal`
Operations receive the normal priority for execution.
Available in iOS 2.0 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityHigh`
Operations receive high priority for execution.
Available in iOS 2.0 and later.
Declared in `NSOperation.h`.

`NSOperationQueuePriorityVeryHigh`
Operations receive very high priority for execution.
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSOperation.h
Companion guide	Concurrency Programming Guide
Related sample code	CryptoExercise

Overview

The `NSOperationQueue` class regulates the execution of a set of `NSOperation` objects. After being added to a queue, an operation remains in that queue until it is explicitly canceled or finishes executing its task. Operations within the queue (but not yet executing) are themselves organized according to priority levels and inter-operation object dependencies and are executed accordingly. An application may create multiple operation queues and submit operations to any of them.

Inter-operation dependencies provide an absolute execution order for operations, even if those operations are located in different operation queues. 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 cannot directly remove an operation from a queue after it has been added. An operation remains in its queue until it reports that it is finished with its task. Finishing its task does not necessarily mean that the operation performed that task to completion. An operation can also be canceled. Canceling an operation object leaves the object in the queue but notifies the object that it should abort its task as quickly as possible. For currently executing operations, this means that the operation object's work code must check the cancellation state, stop what it is doing, and mark itself as finished. For operations that are queued but not yet executing, the queue must still call the operation object's `start` method so that it can processes the cancellation event and mark itself as finished.

Note: In Mac OS X v10.6 and later, canceling an operation causes the operation to ignore any dependencies it may have. This behavior makes it possible for the queue to execute the operation's `start` method as soon as possible. The `start` method, in turn, moves the operation to the finished state so that it can be removed from the queue. In Mac OS X v10.5, a canceled operation does not ignore its dependencies, meaning that those dependencies must complete normally before the canceled operation can run and be removed from the queue.

Operation queues usually provide the threads used to run their operations. In Mac OS X v10.6 and later, operation queues use the `libdispatch` library (also known as Grand Central Dispatch) to initiate the execution of their operations. As a result, operations are always executed on a separate thread, regardless of whether they are designated as concurrent or non-concurrent operations. In Mac OS X v10.5, however, operations are executed on separate threads only if their `isConcurrent` (page 995) method returns `NO`. If that method returns `YES`, the operation object is expected to create its own thread (or start some asynchronous operation); the queue does not provide a thread for it.

Note: In iOS, operation queues do not use Grand Central Dispatch to execute operations. They create separate threads for non-concurrent operations and launch concurrent operations from the current thread. For a discussion of the difference between concurrent and non-concurrent operations and how they are executed, see *NSOperation Class Reference*.

For more information about using operation queues, see *Concurrency Programming Guide*.

KVO-Compliant Properties

The `NSOperationQueue` class is key-value coding (KVC) and key-value observing (KVO) compliant. You can observe these properties as desired to control other parts of your application. The properties you can observe include the following:

- `operations` - read-only property
- `operationCount` - read-only property
- `maxConcurrentOperationCount` - readable and writable property
- `suspended` - readable and writable property
- `name` - readable and writable property

Although you can attach observers to these properties, you should not use Cocoa bindings to bind them to elements of your application's user interface. Code associated with your user interface typically must execute only in your application's main thread. However, KVO notifications associated with an operation queue may occur in any thread.

For more information about key-value observing and how to attach observers to an object, see *Key-Value Observing Programming Guide*.

Multicore Considerations

It is safe to use a single `NSOperationQueue` object from multiple threads without creating additional locks to synchronize access to that object.

Tasks

Managing Operations in the Queue

- `addOperation:` (page 1007)
  Adds the specified operation object to the receiver.
- `addOperations:waitUntilFinished:` (page 1007)
  Adds the specified array of operations to the queue.
- `addOperationWithBlock:` (page 1008)
  Wraps the specified block in an operation object and adds it to the receiver.
- `operations` (page 1010)
  Returns a new array containing the operations currently in the queue.
- `operationCount` (page 1010)
  Returns the number of operations currently in the queue.
- `cancelAllOperations` (page 1008)
  Cancels all queued and executing operations.
- `waitUntilAllOperationsAreFinished` (page 1012)
  Blocks the current thread until all of the receiver's queued and executing operations finish executing.

Managing the Number of Running Operations

- `maxConcurrentOperationCount` (page 1009)
  Returns the maximum number of concurrent operations that the receiver can execute.
- `setMaxConcurrentOperationCount:` (page 1011)
  Sets the maximum number of concurrent operations that the receiver can execute.

Suspending Operations

- `setSuspended:` (page 1012)
  Modifies the execution of pending operations
- `isSuspended` (page 1009)
  Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

Managing the Queue's Name

- `setName:` (page 1011)
  Assigns the specified name to the receiver.
- `name` (page 1009)
  Returns the name of the receiver.

Getting Specific Operation Queues

`+ currentQueue` (page 1006)
Returns the operation queue that launched the current operation.

`+ mainQueue` (page 1006)
Returns the operation queue associated with the main thread.

Class Methods

currentQueue

Returns the operation queue that launched the current operation.

```
+ (id)currentQueue
```

Return Value
The operation queue that started the operation or `nil` if the queue could not be determined.

Discussion
You can use this method from within a running operation object to get a reference to the operation queue that started it. Calling this method from outside the context of a running operation typically results in `nil` being returned.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

mainQueue

Returns the operation queue associated with the main thread.

```
+ (id)mainQueue
```

Return Value
The default operation queue bound to the main thread.

Discussion
The returned queue executes operations serially on the main thread. The main thread's run loop controls the execution times of these operations.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

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. In memory-managed applications, this object is retained by the operation queue. In garbage-collected applications, the queue strongly references the operation object.

Discussion

Once added, the specified *operation* remains in the queue until it finishes executing.

An operation object can be in at most one operation queue at a time and this method throws an `NSInvalidArgumentException` exception if the operation is already in another queue. Similarly, this method throws an `NSInvalidArgumentException` exception if the operation is currently executing or has already finished executing.

Availability

Available in iOS 2.0 and later.

See Also

`cancel` (page 992) (NSOperation)

`isExecuting` (page 995) (NSOperation)

Declared In

`NSOperation.h`

addOperations:waitUntilFinished:

Adds the specified array of operations to the queue.

```
- (void)addOperations:(NSArray *)ops waitUntilFinished:(BOOL)wait
```

Parameters

ops

The array of `NSOperation` objects that you want to add to the receiver.

wait

If `YES`, the current thread is blocked until all of the specified operations finish executing. If `NO`, the operations are added to the queue and control returns immediately to the caller.

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 those error conditions are true for any of the operations in the *ops* parameter.

Once added, the specified *operation* remains in the queue until it its `isFinished` (page 996) method returns `YES`.

Availability
Available in iOS 4.0 and later.

Declared In
NSOperation.h

addOperationWithBlock:

Wraps the specified block in an operation object and adds it to the receiver.

```
- (void)addOperationWithBlock:(void (^)(void))block
```

Parameters

block

The block to execute from the operation object. The block should take no parameters and have no return value.

Discussion
This method adds a single block to the receiver by first wrapping it in an operation object. You should not attempt to get a reference to the newly created operation object or divine its type information.

Once added, the specified *operation* remains in the queue until it its `isFinished` (page 996) method returns `YES`.

Availability
Available in iOS 4.0 and later.

See Also
`cancel` (page 992) (NSOperation)
`isExecuting` (page 995) (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. 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 iOS 2.0 and later.

See Also
`cancel` (page 992) (`NSOperation`)

Declared In
NSOperation.h

isSuspended

Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

```
- (BOOL)isSuspended
```

Return Value
`NO` if operations are being scheduled for execution; otherwise, `YES`.

Discussion
If you want to know when the queue's suspended state changes, configure a KVO observer to observe the `suspended` key path of the operation queue.

Availability
Available in iOS 2.0 and later.

See Also
`- setSuspended:` (page 1012)

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. If no value has been explicitly set, this method returns `NSOperationQueueDefaultMaxConcurrentOperationCount` by default.

Availability
Available in iOS 2.0 and later.

See Also
`- setMaxConcurrentOperationCount:` (page 1011)

Declared In
`NSOperation.h`

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Discussion
The default value of this string is "`NSOperationQueue <id>`", where <*id*> is the memory address of the operation queue. If you want to know when a queue's name changes, configure a KVO observer to observe the `name` key path of the operation queue.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

operationCount

Returns the number of operations currently in the queue.

```
- (NSUInteger)operationCount
```

Return Value
The number of operations in the queue.

Discussion
The value returned by this method reflects the instantaneous number of objects in the queue and changes as operations are completed. As a result, by the time you use the returned value, the actual number of operations may be different. You should therefore use this value only for approximate guidance and should not rely on it for object enumerations or other precise calculations.

Availability
Available in iOS 4.0 and later.

Declared In
`NSOperation.h`

operations

Returns a new array containing the operations currently in the queue.

```
- (NSArray *)operations
```

Return Value
A new array object containing the `NSOperation` objects in the order in which they were added to the queue.

Discussion
You can use this method to access the operations queued at any given moment. Operations remain queued until they finish their task. Therefore, the returned array may contain operations that are either executing or waiting to be executed. The list may also contain operations that were executing when the array was initially created but have subsequently finished.

Availability
Available in iOS 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.

Note: Setting the maximum number of operations to 1 effectively creates a serial queue for processing operations.

Availability

Available in iOS 2.0 and later.

See Also

`- maxConcurrentOperationCount` (page 1009)

Declared In

`NSOperation.h`

setName:

Assigns the specified name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

newName

The new name to associate with the receiver.

Discussion

Names provide a way for you to identify your operation queues at run time. Tools may also use this name to provide additional context during debugging or analysis of your code.

Availability

Available in iOS 4.0 and later.

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 resumes the execution of operations. Suspending a queue prevents that queue from starting additional operations. In other words, operations that are in the queue (or added to the queue later) and are not yet executing are prevented from starting until the queue is resumed. Suspending a queue does not stop operations that are already running.

Operations are removed from the queue only when they finish executing. However, in order to finish executing, an operation must first be started. Because a suspended queue does not start any new operations, it does not remove any operations (including cancelled operations) that are currently queued and not executing.

Availability

Available in iOS 2.0 and later.

See Also

- isSuspended (page 1009)

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 queued operations to finish executing. While the current 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.

If there are no operations in the queue, this method returns immediately.

Availability

Available in iOS 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 iOS 2.0 and later.

Declared in `NSOperation.h`.

Declared In

`NSOperation.h`

NSOrthography Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSOrthography.h

Overview

The `NSOrthography` class describes the linguistic content of a piece of text, typically used for the purposes of spelling and grammar checking.

An `NSOrthography` instance describes:

- Which scripts the text contains.
- A dominant language and possibly other languages for each of these scripts.
- A dominant script and language for the text as a whole.

Scripts are uniformly described by standard four-letter tags (`Latn`, `Grek`, `Cyrl`, etc.) with the supertags `Jpan` and `Kore` typically used for Japanese and Korean text, `Hans` and `Hant` for Chinese text; the tag `Zyyy` is used if a specific script cannot be identified. See *Internationalization Programming Topics* for more information on internationalization.

Languages are uniformly described by BCP-47 tags , preferably in canonical form; the tag `und` is used if a specific language cannot be determined.

Subclassing Notes

Methods to Override

The `dominantScript` (page 1017) and `languageMap` (page 1017) properties are the primitive values that a subclass must implement. The properties are set using the `initWithDominantScript:languageMap:` (page 1019) or `orthographyWithDominantScript:languageMap:` (page 1018).

Tasks

Creating Instances of NSOrthography

`+ orthographyWithDominantScript:languageMap:` (page 1018)
Creates and returns an orthography instance with the specified dominant script and language map.

`- initWithDominantScript:languageMap:` (page 1019)
Creates and returns an orthography instance with the specified dominant script and language map.

Defining the Language Map

`dominantScript` (page 1017) *property*
The dominant script for the text. (read-only)

`languageMap` (page 1017) *property*
A dictionary that map script tags to arrays of language tags. (read-only)

Managing Languages and Scripts

`- languagesForScript:` (page 1019)
Returns the list of languages for the specified script.

`- dominantLanguageForScript:` (page 1018)
Returns the dominant language for the specified script.

`allLanguages` (page 1016) *property*
Returns an array containing all the languages appearing in the values of the language map. (read-only)

`allScripts` (page 1017) *property*
Returns an array containing all the scripts appearing as keys in the language map. (read-only)

`dominantLanguage` (page 1017) *property*
Returns the first language in the list of languages for the dominant script. (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

allLanguages

Returns an array containing all the languages appearing in the values of the language map. (read-only)

```
@property(readonly) NSArray *allLanguages
```

Availability
Available in iOS 4.0 and later.

Declared In
NSOrthography.h

allScripts

Returns an array containing all the scripts appearing as keys in the language map. (read-only)

```
@property(readonly) NSArray *allScripts
```

Availability
Available in iOS 4.0 and later.

Declared In
NSOrthography.h

dominantLanguage

Returns the first language in the list of languages for the dominant script. (read-only)

```
@property(readonly) NSString *dominantLanguage
```

Availability
Available in iOS 4.0 and later.

Declared In
NSOrthography.h

dominantScript

The dominant script for the text. (read-only)

```
@property(readonly) NSString *dominantScript
```

Discussion
The dominant script should be a script tag, such as `Latn`, `Cyrl`, etc.

Availability
Available in iOS 4.0 and later.

See Also
`@property languageMap` (page 1017)

Declared In
NSOrthography.h

languageMap

A dictionary that map script tags to arrays of language tags. (read-only)

```
@property(readonly) NSDictionary *languageMap
```

Discussion
The dictionary's keys are script tags (such as `Latn`, `Cyrl`, and so forth) and whose values are arrays of language tags (such as `en`, `fr`, `de`, etc.)

Availability
Available in iOS 4.0 and later.

See Also
`@property dominantScript` (page 1017)

Declared In
`NSOrthography.h`

Class Methods

orthographyWithDominantScript:languageMap:

Creates and returns an orthography instance with the specified dominant script and language map.

```
+ (id)orthographyWithDominantScript:(NSString *)scriptlanguageMap:(NSDictionary
    *)map
```

Parameters

script
The dominant script.

map
A dictionary containing the language map.

Return Value
An initialized orthography object for the specified script and language map.

Availability
Available in iOS 4.0 and later.

See Also
`- initWithDominantScript:languageMap:` (page 1019)

Declared In
`NSOrthography.h`

Instance Methods

dominantLanguageForScript:

Returns the dominant language for the specified script.

```
- (NSString *)dominantLanguageForScript:(NSString *)script
```

Parameters

script

The script.

Return Value

A string containing the dominant language

Availability

Available in iOS 4.0 and later.

Declared In

NSOrthography.h

initWithDominantScript:languageMap:

Creates and returns an orthography instance with the specified dominant script and language map.

```
- (id)initWithDominantScript:(NSString *)scriptlanguageMap:(NSDictionary *)map
```

Parameters

script

The dominant script.

map

A dictionary containing the language map.

Return Value

An initialized orthography object for the specified script and language map.

Availability

Available in iOS 4.0 and later.

See Also

+ orthographyWithDominantScript:languageMap: (page 1018)

Declared In

NSOrthography.h

languagesForScript:

Returns the list of languages for the specified script.

```
- (NSArray *)languagesForScript:(NSString *)script
```

Parameters

script

The script.

Return Value

An array of strings containing the languages.

Availability

Available in iOS 4.0 and later.

See Also

- `dominantLanguageForScript:` (page 1018)
- `@property allLanguages` (page 1016)

Declared In

`NSOrthography.h`

NSOutputStream Class Reference

Inherits from	NSStream : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CryptoExercise WiTap

Overview

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 1027)

  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 1025)

  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 1024)
Creates and returns an initialized output stream that will write stream data to memory.

`+ outputStreamToBuffer:capacity:` (page 1022)
Creates and returns an initialized output stream that can write to a provided buffer.

`+ outputStreamToFileAtPath:append:` (page 1023)
Creates and returns an initialized output stream for writing to a specified file.

`+ outputStreamWithURL:append:` (page 1024)
Creates and returns an initialized output stream for writing to a specified URL.

`- initToMemory` (page 1026)
Returns an initialized output stream that will write to memory.

`- initToBuffer:capacity:` (page 1025)
Returns an initialized output stream that can write to a provided buffer.

`- initToFileAtPath:append:` (page 1026)
Returns an initialized output stream for writing to a specified file.

`- initWithURL:append:` (page 1027)
Returns an initialized output stream for writing to a specified URL.

Using Streams

`- hasSpaceAvailable` (page 1025)
Returns whether the receiver can be written to.

`- write:maxLength:` (page 1027)
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 1173) will return `NSStreamStatusAtEnd`.

Availability

Available in iOS 2.0 and later.

See Also

`+ outputStreamToMemory` (page 1024)

`+ outputStreamToFileAtPath:append:` (page 1023)

`- initToBuffer:capacity:` (page 1025)

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 iOS 2.0 and later.

See Also

`+ outputStreamToMemory` (page 1024)

`+ outputStreamToBuffer:capacity:` (page 1022)

`- initToFileAtPath:append:` (page 1026)

- initWithURL:append: (page 1027)

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 1170) to the receiver with an argument of NSStreamDataWrittenToMemoryStreamKey.

Availability
Available in iOS 2.0 and later.

See Also
+ outputStreamToBuffer:capacity: (page 1022)
+ outputStreamToFileAtPath:append: (page 1023)
- initToMemory (page 1026)

Declared In
NSStream.h

outputStreamWithURL:append:

Creates and returns an initialized output stream for writing to a specified URL.

```
+ (id)outputStreamWithURL:(NSURL *)url append:(BOOL)shouldAppend
```

Parameters

url
The URL 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 *url*.

Discussion
The stream must be opened before it can be used.

Availability
Available in iOS 4.0 and later.

See Also
+ outputStreamToMemory (page 1024)
+ outputStreamToBuffer:capacity: (page 1022)

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 iOS 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 1173) will return NSStreamStatusAtEnd.

Availability
Available in iOS 2.0 and later.

See Also
- initToMemory (page 1026)

- initToFileAtPath:append: (page 1026)
+ outputStreamToBuffer:capacity: (page 1022)

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 iOS 2.0 and later.

See Also
- initToMemory (page 1026)
- initToBuffer:capacity: (page 1025)
+ outputStreamToFileAtPath:append: (page 1023)
+ outputStreamWithURL:append: (page 1024)

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 1170).

Availability
Available in iOS 2.0 and later.

See Also
- `initToBuffer:capacity:` (page 1025)
- `initToFileAtPath:append:` (page 1026)
+ `outputStreamToMemory` (page 1024)

Declared In
`NSStream.h`

initWithURL:append:

Returns an initialized output stream for writing to a specified URL.

```
- (id)initWithURL:(NSURL *)url append:(BOOL)shouldAppend
```

Parameters

url

The URL 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 *url*.

Discussion
The stream must be opened before it can be used.

Availability
Available in iOS 4.0 and later.

See Also
- `initToMemory` (page 1026)
- `initToBuffer:capacity:` (page 1025)

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. More information about the error can be obtained with `streamError` (page 1173). If the receiver is a fixed-length stream and has reached its capacity, `0` is returned.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSFileHandle.h
Companion guide	Interacting with the Operating System

Overview

`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 1031)
Returns an initialized `NSPipe` object.

`+ pipe` (page 1030)
Returns an `NSPipe` object.

Getting the File Handles for a Pipe

`- fileHandleForReading` (page 1030)
Returns the receiver's read file handle.

`- fileHandleForWriting` (page 1030)
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 iOS 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 478), `readDataToEndOfFile` (page 482), and `readDataOfLength:` (page 481).

You don't need to send `closeFile` (page 479) to this object or explicitly release the object after you have finished using it.

Availability
Available in iOS 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 487) method. When you are finished writing data to this object, send it a `closeFile` (page 479) 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 iOS 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 iOS 2.0 and later.

See Also
`+ pipe` (page 1030)

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 iOS 2.0 and later.
Declared in	Foundation/NSPort.h
Companion guides	Distributed Objects Programming Topics Threading Programming Guide

Overview

`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 1611) or `handlePortMessage:` (page 1641) message. The delegate should implement only one of these methods to process the incoming message in whatever form desired. `handleMachMessage:` (page 1611) provides a message as a raw Mach message beginning with a `msg_header_t` structure. `handlePortMessage:` (page 1641) 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 1035) or `port` (page 1035), 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

Tasks

Creating Instances

`+ allocWithZone:` (page 1035)
Returns an instance of the `NSMachPort` class.

`+ port` (page 1035)
Creates and returns a new `NSPort` object capable of both sending and receiving messages.

Validation

`- invalidate` (page 1036)
Marks the receiver as invalid and posts an `NSPortDidBecomeInvalidNotification` (page 1040) to the default notification center.

`- isValid` (page 1037)
Returns a Boolean value that indicates whether the receiver is valid.

Setting the Delegate

`- setDelegate:` (page 1039)
Sets the receiver's delegate to a given object.

`- delegate` (page 1036)
Returns the receiver's delegate.

Setting Information

`- sendBeforeDate:components:from:reserved:` (page 1038)
This method is provided for subclasses that have custom types of `NSPort`.

- `sendBeforeDate:msgid:components:from:reserved:` (page 1039)
  This method is provided for subclasses that have custom types of `NSPort`.
- `reservedSpaceLength` (page 1037)
  Returns the number of bytes of space reserved by the receiver for sending data.

Port Monitoring

- `removeFromRunLoop:forMode:` (page 1037)
  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 1038)
  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.

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 iOS 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 iOS 2.0 and later.

See Also
`+ allocWithZone:` (page 1035)

Declared In
`NSPort.h`

Instance Methods

delegate

Returns the receiver's delegate.

```
- (id < NSPortDelegate >)delegate
```

Return Value
The receiver's delegate.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 1039)

Declared In
`NSPort.h`

invalidate

Marks the receiver as invalid and posts an `NSPortDidBecomeInvalidNotification` (page 1040) 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 iOS 2.0 and later.

See Also
`- isValid` (page 1037)

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 iOS 2.0 and later.

See Also
`- invalidate` (page 1036)

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 iOS 2.0 and later.

See Also
`- scheduleInRunLoop:forMode:` (page 1038)

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 iOS 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 iOS 2.0 and later.

See Also
`- removeFromRunLoop:forMode:` (page 1037)

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 iOS 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 iOS 2.0 and later.

Declared In
`NSPort.h`

setDelegate:

Sets the receiver's delegate to a given object.

```
- (void)setDelegate:(id < NSPortDelegate >)anObject
```

Parameters

anObject
The delegate for the receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- delegate` (page 1036)

Declared In
`NSPort.h`

Notifications

NSPortDidBecomeInvalidNotification

Posted from the `invalidate` (page 1036) 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 iOS 2.0 and later.

Declared In
`NSPort.h`

NSPredicate Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	Foundation/NSPredicate.h
Companion guide	Predicate Programming Guide
Related sample code	ToolbarSearch

Overview

The `NSPredicate` class is used to define logical conditions used to constrain a search either for a fetch or for in-memory filtering.

You use predicates to represent logical conditions, used for describing objects in persistent stores and in-memory filtering of objects. Although it is common to create predicates directly from instances of `NSComparisonPredicate`, `NSCompoundPredicate`, and `NSExpression`, you often create predicates from a format string which is parsed by the class methods on `NSPredicate`. Examples of predicate format strings include:

- Simple comparisons, such as `grade == "7"` or `firstName like "Shaffiq"`
- Case and diacritic insensitive lookups, such as `name contains[cd] "itroen"`
- Logical operations, such as `(firstName like "Mark") OR (lastName like "Adderley")`
- In Mac OS X v10.5 and later, you can create between predicates such as `date between {$YESTERDAY, $TOMORROW}`.

You can create predicates for relationships, such as:

- `group.name like "work*"`
- `ALL children.age > 12`
- `ANY children.age > 12`

You can create predicates for operations, such as `@sum.items.price < 1000`. For a complete syntax reference, refer to the *Predicate Programming Guide*.

You can also create predicates that include variables, so that the predicate can be pre-defined before substituting concrete values at runtime. In Mac OS X v10.4, for predicates that use variables, evaluation is a two step process (see `predicateWithSubstitutionVariables:` (page 1047) and `evaluateWithObject:` (page 1045)). In Mac OS X v10.5 and later, you can use `evaluateWithObject:substitutionVariables:` (page 1046), which combines these steps.

Tasks

Creating a Predicate

`+ predicateWithFormat:` (page 1043)
Creates and returns a new predicate formed by creating a new string with a given format and parsing the result.

`+ predicateWithFormat:argumentArray:` (page 1044)
Creates and returns a new predicate by substituting the values in a given array into a format string and parsing the result.

`+ predicateWithFormat:arguments:` (page 1044)
Creates and returns a new predicate by substituting the values in an argument list into a format string and parsing the result.

`- predicateWithSubstitutionVariables:` (page 1047)
Returns a copy of the receiver with the receiver's variables substituted by values specified in a given substitution variables dictionary.

`+ predicateWithValue:` (page 1045)
Creates and returns a predicate that always evaluates to a given value.

`+ predicateWithBlock:` (page 1043)
Creates and returns a predicate that evaluates using a specified block object and bindings dictionary.

Evaluating a Predicate

`- evaluateWithObject:` (page 1045)
Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver.

`- evaluateWithObject:substitutionVariables:` (page 1046)
Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver after substituting in the values in a given variables dictionary.

Getting a String Representation

`- predicateFormat` (page 1046)
Returns the receiver's format string.

Class Methods

predicateWithBlock:

Creates and returns a predicate that evaluates using a specified block object and bindings dictionary.

```
+ (NSPredicate *)predicateWithBlock:(BOOL (^)(id evaluatedObject, NSDictionary
    *bindings))block
```

Parameters

block

The block is applied to the object to be evaluated.

The block takes two arguments:

evaluatedObject

The object to be evaluated.

bindings

The substitution variables dictionary. The dictionary must contain key-value pairs for all variables in the receiver.

The block returns `YES` if the *evaluatedObject* evaluates to true, otherwise `NO`.

Return Value

A new predicate by that evaluates objects using *block*.

Special Considerations

In Mac OS X v10.6, Core Data supports this method in the in-memory and atomic stores, but not in the SQLite-based store.

Availability

Available in iOS 4.0 and later.

Declared In

`NSPredicate.h`

predicateWithFormat:

Creates and returns a new predicate formed by creating a new string with a given format and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)format, ...
```

Parameters

format

The format string for the new predicate.

...

A comma-separated list of arguments to substitute into *format*.

Return Value

A new predicate formed by creating a new string with *format* and parsing the result.

Discussion
For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability
Available in iOS 3.0 and later.

Related Sample Code
ToolbarSearch

Declared In
NSPredicate.h

predicateWithFormat:argumentArray:

Creates and returns a new predicate by substituting the values in a given array into a format string and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)predicateFormat
    argumentArray:(NSArray *)arguments
```

Parameters

predicateFormat
The format string for the new predicate.

arguments
The arguments to substitute into *predicateFormat*. Values are substituted into *predicateFormat* in the order they appear in the array.

Return Value
A new predicate by substituting the values in *arguments* into *predicateFormat*, and parsing the result.

Discussion
For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

predicateWithFormat:arguments:

Creates and returns a new predicate by substituting the values in an argument list into a format string and parsing the result.

```
+ (NSPredicate *)predicateWithFormat:(NSString *)format arguments:(va_list)argList
```

Parameters

format
The format string for the new predicate.

argList
The arguments to substitute into *predicateFormat*. Values are substituted into *predicateFormat* in the order they appear in the argument list.

Return Value
A new predicate by substituting the values in *argList* into *predicateFormat* and parsing the result.

Discussion
For details of the format of the format string and of limitations on variable substitution, see Predicate Format String Syntax.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

predicateWithValue:

Creates and returns a predicate that always evaluates to a given value.

```
+ (NSPredicate *)predicateWithValue:(BOOL)value
```

Parameters
value
The value to which the new predicate should evaluate.

Return Value
A predicate that always evaluates to *value*.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

Instance Methods

evaluateWithObject:

Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver.

```
- (BOOL)evaluateWithObject:(id)object
```

Parameters
object
The object against which to evaluate the receiver.

Return Value
YES if *object* matches the conditions specified by the receiver, otherwise NO.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

evaluateWithObject:substitutionVariables:

Returns a Boolean value that indicates whether a given object matches the conditions specified by the receiver after substituting in the values in a given variables dictionary.

```
- (BOOL)evaluateWithObject:(id)object substitutionVariables:(NSDictionary *)variables
```

Parameters

object

The object against which to evaluate the receiver.

variables

The substitution variables dictionary. The dictionary must contain key-value pairs for all variables in the receiver.

Return Value
YES if *object* matches the conditions specified by the receiver after substituting in the values in *variables* for any replacement tokens, otherwise NO.

Discussion
This method returns the same result as the two step process of first invoking predicateWithSubstitutionVariables: (page 1047) on the receiver and then invoking evaluateWithObject: (page 1045) on the returned predicate. This method is optimized for situations which require repeatedly evaluating a predicate with substitution variables with different variable substitutions.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

predicateFormat

Returns the receiver's format string.

```
- (NSString *)predicateFormat
```

Return Value
The receiver's format string.

Special Considerations

The string returned by this method is not guaranteed to be the same as a string used to create the predicate using predicateWithFormat: etc. You cannot use this method to create a persistent representation of a predicate that you could use to recreate the original predicate. If you need a persistent representation of a predicate, create an archive (NSPredicate adopts the NSCoding protocol).

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

predicateWithSubstitutionVariables:

Returns a copy of the receiver with the receiver's variables substituted by values specified in a given substitution variables dictionary.

```
- (NSPredicate *)predicateWithSubstitutionVariables:(NSDictionary *)variables
```

Parameters

variables

The substitution variables dictionary. The dictionary must contain key-value pairs for all variables in the receiver.

Return Value
A copy of the receiver with the receiver's variables substituted by values specified in *variables*.

Discussion
The receiver itself is not modified by this method, so you can reuse it for any number of substitutions.

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.h

NSProcessInfo Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSProcessinfo.h
Companion guide	Interacting with the Operating System

Overview

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 1051) 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 1054) 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.

Sudden Termination

Mac OS X v10.6 includes a new mechanism that allows the system to log out or shut down more quickly by, whenever possible, killing applications instead of requesting that they quit themselves.

Your application can enable this capability on a global basis and then manually override its availability during actions that could cause data corruption or a poor user experience by allowing sudden termination. Alternately, your application can just manually enable and disable this functionality.

The methods `enableSuddenTermination` and `disableSuddenTermination` decrement or increment, respectively, a counter whose value is `1` when the process is first created. When the counter's value is 0 the application is considered to be safely killable and may be killed by the system without any notification or event being sent to the process first.

Your application can support sudden termination upon launch by adding a key to the application's Info.plist. If the `NSSupportsSuddenTermination` key exists in the Info.plist and has a value of `YES`, it is the equivalent of calling `enableSuddenTermination` during your application launch. This renders the application process killable right away. You can still override this behavior by invoking `disableSuddenTermination`.

Typically, you will disable sudden termination whenever your application defers work that must be done before the application terminates. If, for example, your application defers writing data to disk, and sudden termination is enabled, you should bracket the sensitive operations with a call to `disableSuddenTermination`, perform the necessary operations, and then send a balancing `enableSuddenTermination` message.

In agents or daemon executables that don't depend on Application Kit you can manually invoke `enableSuddenTermination` right away. You can then use the enable and disable methods whenever the process has work it must do before it terminates.

Some Application Kit functionality automatically disables sudden termination on a temporary basis to ensure data integrity.

- `NSUserDefaults` temporarily disables sudden termination to prevent process killing between the time at which a default has been set and the time at which the preferences file including that default has been written to disk.
- `NSDocument` temporarily disables sudden termination to prevent process killing between the time at which the user has made a change to a document and the time at which the user's change has been written to disk.

Debugging tip: You can determine the value of the sudden termination using the following gdb command.

```
print (long)[[NSClassFromString(@"NSProcessInfo") processInfo]
_suddenTerminationDisablingCount
```

Do not attempt to invoke or override `suddenTerminationDisablingCount` (a private method) in your application. It is there just for this debugging purpose, and may disappear at any time.

Tasks

Getting the Process Information Agent

`+ processInfo` (page 1051)
Returns the process information agent for the process.

Accessing Process Information

- `arguments` (page 1052)
  Returns the command-line arguments for the process.
- `environment` (page 1053)
  Returns the variable names and their values in the environment from which the process was launched.
- `processIdentifier` (page 1055)
  Returns the identifier of the process.
- `globallyUniqueString` (page 1053)
  Returns a global unique identifier for the process.
- `processName` (page 1055)
  Returns the name of the process.
- `setProcessName:` (page 1056)
  Sets the name of the process.

Getting Host Information

- `hostName` (page 1053)
  Returns the name of the host computer.
- `operatingSystem` (page 1054)
  Returns a constant to indicate the operating system on which the process is executing.
- `operatingSystemName` (page 1054)
  Returns a string containing the name of the operating system on which the process is executing.
- `operatingSystemVersionString` (page 1054)
  Returns a string containing the version of the operating system on which the process is executing.

Getting Computer Information

- `physicalMemory` (page 1054)
  Provides the amount of physical memory on the computer.
- `processorCount` (page 1056)
  Provides the number of processing cores available on the computer.
- `activeProcessorCount` (page 1052)
  Provides the number of active processing cores available on the computer.
- `systemUptime` (page 1056)
  Returns how long it has been since the computer has been restarted.

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 1049) object is created the first time this method is invoked, and that same object is returned on each subsequent invocation.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`- processorCount` (page 1056)

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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
- processName (page 1055)

Declared In
NSProcessInfo.h

hostName

Returns the name of the host computer.

```
- (NSString *)hostName
```

Return Value
Host name of the computer.

Availability
Available in iOS 2.0 and later.

Declared In
NSProcessInfo.h

operatingSystem

Returns a constant to indicate the operating system on which the process is executing.

```
- (NSUInteger)operatingSystem
```

Return Value
Operating system identifier. See "Constants" (page 1057) for a list of possible values. In Mac OS X, it's `NSMACHOperatingSystem`.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
- processName (page 1055)

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 iOS 2.0 and later.

See Also
- processIdentifier (page 1055)
- setProcessName: (page 1056)

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 iOS 2.0 and later.

See Also
- `activeProcessorCount` (page 1052)

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 iOS 2.0 and later.

See Also
- `processName` (page 1055)

Declared In
`NSProcessInfo.h`

systemUptime

Returns how long it has been since the computer has been restarted.

```
- (NSTimeInterval)systemUptime
```

Return Value
An `NSTimeInterval` (page 1752) indicating how long since the computer has been restarted.

Availability
Available in iOS 4.0 and later.

Declared In
NSProcessInfo.h

Constants

NSProcessInfo—Operating Systems

The following constants are provided by the NSProcessInfo class as return values for operatingSystem (page 1054).

```
enum {
   NSWindowsNTOperatingSystem = 1,
   NSWindows95OperatingSystem,
   NSSolarisOperatingSystem,
   NSHPUXOperatingSystem,
   NSMACHOperatingSystem,
   NSSunOSOperatingSystem,
   NSOSF1OperatingSystem
};
```

Constants

NSHPUXOperatingSystem
Indicates the HP UX operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

NSMACHOperatingSystem
Indicates the Mac OS X operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

NSOSF1OperatingSystem
Indicates the OSF/1 operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

NSSolarisOperatingSystem
Indicates the Solaris operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

NSSunOSOperatingSystem
Indicates the Sun OS operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

NSWindows95OperatingSystem
Indicates the Windows 95 operating system.
Available in iOS 2.0 and later.
Declared in NSProcessInfo.h.

`NSWindowsNTOperatingSystem`

Indicates the Windows NT operating system.

Available in iOS 2.0 and later.

Declared in `NSProcessInfo.h`.

NSPropertyListSerialization Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSPropertyList.h
Companion guides	Archives and Serializations Programming Guide Property List Programming Guide
Related sample code	CryptoExercise ScrollViewSuite

Overview

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 1060)
Returns an `NSData` object containing a given property list in a specified format.

`+ dataWithPropertyList:format:options:error:` (page 1061)
Returns an `NSData` object containing a given property list in a specified format.

`+ writePropertyList:toStream:format:options:error:` (page 1064)
Writes the specified property list to the specified stream.

Deserializing a Property List

`+ propertyListFromData:mutabilityOption:format:errorDescription:` (page 1062)
Returns a property list object corresponding to the representation in a given `NSData` object.

`+ propertyListWithData:options:format:error:` (page 1063)
Creates and returns a property list from the specified data.

`+ propertyListWithStream:options:format:error:` (page 1063)
Creates and returns a property list by reading from the specified stream.

Validating a Property List

`+ propertyList:isValidForFormat:` (page 1061)
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 1065).

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*.

Discussion
Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Important: This method is obsolete and will be deprecated soon. Use `dataWithPropertyList:format:options:error:` (page 1061) instead.

Availability
Available in iOS 2.0 and later.

See Also
`+ dataWithPropertyList:format:options:error:` (page 1061)

Related Sample Code
CryptoExercise

Declared In
`NSPropertyList.h`

dataWithPropertyList:format:options:error:

Returns an `NSData` object containing a given property list in a specified format.

```
+ (NSData *)dataWithPropertyList:(id)plist format:(NSPropertyListFormat)format
    options:(NSPropertyListWriteOptions)opt error:(NSError **)error
```

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. Passing `nil` for this value will cause an exception to be raised.

format
A property list format. Possible values for *format* are described in NSPropertyListFormat (page 1065).

opt
The opt parameter is currently unused and should be set to 0.

error
If the method does not complete successfully, upon return contains an NSError object that describes the problem.

Return Value
An `NSData` object containing *plist* in the format specified by *format*.

Availability
Available in iOS 4.0 and later.

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 1065).

Return Value
`YES` if *plist* is a valid property list in format *format*, otherwise `NO`.

Availability
Available in iOS 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

The opt parameter is currently unused and should be set to 0.

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 1065).

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.

Discussion
Unlike the normal memory management rules for Cocoa, strings returned in *errorString* need to be released by the caller.

Important: This method is obsolete and will be deprecated soon. Use propertyListWithData:options:format:error: (page 1063) instead.

Availability
Available in iOS 2.0 and later.

See Also
+ propertyListWithData:options:format:error: (page 1063)

Related Sample Code
CryptoExercise
ScrollViewSuite

Declared In
NSPropertyList.h

propertyListWithData:options:format:error:

Creates and returns a property list from the specified data.

```
+ (id)propertyListWithData:(NSData *)data options:(NSPropertyListReadOptions)opt
    format:(NSPropertyListFormat *)format error:(NSError **)error
```

Parameters

data

A data object containing a serialized property list.

opt

The opt parameter is currently unused and should be set to 0.

format

A property list format. Possible values for *format* are described in NSPropertyListFormat (page 1065).

error

If the method does not complete successfully, upon return contains an NSError object that describes the problem.

Return Value

A property list object corresponding to the representation in *data*. If data is not in a supported format, returns `nil`.

Availability

Available in iOS 4.0 and later.

Declared In

`NSPropertyList.h`

propertyListWithStream:options:format:error:

Creates and returns a property list by reading from the specified stream.

```
+ (id)propertyListWithStream:(NSInputStream *)stream
    options:(NSPropertyListReadOptions)opt format:(NSPropertyListFormat *)format
    error:(NSError **)error
```

Parameters

stream

An NSStream. The stream should be open and configured for reading.

opt

The opt parameter should be set to on one of the "NSPropertyListMutabilityOptions" (page 1064) options.

format

A property list format. Possible values for *format* are described in NSPropertyListFormat (page 1065).

error

If the method does not complete successfully, upon return contains an NSError object that describes the problem.

Return Value

A property list object corresponding to the representation in *data*. If data is not in a supported format, returns `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
NSPropertyList.h

writePropertyList:toStream:format:options:error:

Writes the specified property list to the specified stream.

```
+ (NSInteger)writePropertyList:(id)plist toStream:(NSOutputStream *)stream
    format:(NSPropertyListFormat)format options:(NSPropertyListWriteOptions)opt
    error:(NSError **)error
```

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.

stream
An NSStream. The stream should be open and configured for writing.

format
A property list format. Possible values for *format* are described in NSPropertyListFormat (page 1065).

opt
The opt parameter is currently unused and should be set to 0.

error
If the method does not complete successfully, upon return contains an NSError object that describes the problem.

Return Value
Returns the number of bytes written to the stream. If the value is 0 an error occurred.

Availability
Available in iOS 4.0 and later.

Declared In
NSPropertyList.h

Constants

NSPropertyListMutabilityOptions

These constants specify mutability options in property lists.

```
enum {
    NSPropertyListImmutable = kCFPropertyListImmutable,
    NSPropertyListMutableContainers = kCFPropertyListMutableContainers,
    NSPropertyListMutableContainersAndLeaves =
kCFPropertyListMutableContainersAndLeaves
};
typedef NSUInteger NSPropertyListMutabilityOptions;
```

Constants

`NSPropertyListImmutable`

Causes the returned property list to contain immutable objects.

Available in iOS 2.0 and later.

Declared in `NSPropertyList.h`.

`NSPropertyListMutableContainers`

Causes the returned property list to have mutable containers but immutable leaves.

Available in iOS 2.0 and later.

Declared in `NSPropertyList.h`.

`NSPropertyListMutableContainersAndLeaves`

Causes the returned property list to have mutable containers and leaves.

Available in iOS 2.0 and later.

Declared in `NSPropertyList.h`.

NSPropertyListFormat

These constants are used to specify a property list serialization format.

```
enum {
    NSPropertyListOpenStepFormat = kCFPropertyListOpenStepFormat,
    NSPropertyListXMLFormat_v1_0 = kCFPropertyListXMLFormat_v1_0,
    NSPropertyListBinaryFormat_v1_0 = kCFPropertyListBinaryFormat_v1_0
}; NSPropertyListFormat;
typedef NSUInteger 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 iOS 2.0 and later.

Declared in `NSPropertyList.h`.

`NSPropertyListXMLFormat_v1_0`

Specifies the XML property list format.

Available in iOS 2.0 and later.

Declared in `NSPropertyList.h`.

`NSPropertyListBinaryFormat_v1_0`
: Specifies the binary property list format.

  Available in iOS 2.0 and later.

  Declared in `NSPropertyList.h`.

NSPropertyListReadOptions

The read options are not currently implemented and the value should be set to `0`.

```
typedef NSUInteger NSPropertyListReadOptions;
```

Availability
Available in iOS 4.0 and later.

Declared In
`NSPropertyList.h`

NSPropertyListWriteOptions

The write options should be set to one of the "NSPropertyListMutabilityOptions" (page 1064) constants.

```
typedef NSUInteger NSPropertyListWriteOptions;
```

Availability
Available in iOS 4.0 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 iOS 2.0 and later.
Declared in	Foundation/NSProxy.h
Companion guide	Distributed Objects Programming Topics

Overview

`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 1071) and `methodSignatureForSelector:` (page 1072) methods to handle messages that it doesn't implement itself. A subclass's implementation of `forwardInvocation:` (page 1071) 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 1072) 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 1629)
- `class` (page 1630)
- `conformsToProtocol:` (page 1630)

- description (page 1631)
- hash (page 1631)
- isEqual: (page 1632)
- isKindOfClass: (page 1632)
- isMemberOfClass: (page 1633)
- isProxy (page 1634)
- performSelector: (page 1634)
- performSelector:withObject: (page 1635)
- performSelector:withObject:withObject: (page 1635)
- release (page 1636)
- respondsToSelector: (page 1637)
- retain (page 1638)
- retainCount (page 1638)
- self (page 1639)
- superclass (page 1640)
- zone (page 1640)

Tasks

Creating Instances

+ alloc (page 1069)
Returns a new instance of the receiving class

+ allocWithZone: (page 1069)
Returns a new instance of the receiving class

Deallocating Instances

- dealloc (page 1070)
Deallocates the memory occupied by the receiver.

Finalizing an Object

- finalize (page 1071)
The garbage collector invokes this method on the receiver before disposing of the memory it uses.

Handling Unimplemented Methods

- forwardInvocation: (page 1071)
Passes a given invocation to the real object the proxy represents.

- `methodSignatureForSelector:` (page 1072)
  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 1070)
  Returns a Boolean value that indicates whether the receiving class responds to a given selector.

Describing a Proxy Class or Object

+ `class` (page 1070)
  Returns `self` (the class object).

- `description` (page 1071)
  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 iOS 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 950) class method.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`class` (page 952) (`NSObject`)
`class` (page 1630) (`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 iOS 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 966) instance method.

Availability
Available in iOS 2.0 and later.

See Also
- `finalize` (page 1071)

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 iOS 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 968) instance method. Note that a `finalize` method must be thread-safe.

Availability
Available in iOS 2.0 and later.

See Also
- `dealloc` (page 1070)

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 iOS 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 iOS 2.0 and later.

See Also
`methodSignatureForSelector:` (page 974) (`NSObject`)

Declared In
`NSProxy.h`

NSPurgeableData Class Reference

Inherits from	NSMutableData : NSData : NSObject
Conforms to	NSDiscardableContent NSCoding (NSData) NSCopying (NSData) NSMutableCopying (NSData) NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	NSData.h

Overview

You should use the `NSPurgeableData` class when you have objects with bytes that can be discarded when no longer needed. Purging these bytes may be advantageous for your system, because doing so frees up memory needed by other applications. The `NSPurgeableData` class provides a default implementation of the `NSDiscardableContent` protocol, from which it inherits its interface.

`NSPurgeableData` objects inherit their creation methods from their superclass, `NSMutableData`. All `NSPurgeableData` objects begin "accessed" to ensure that they are not instantly discarded (see `NSDiscardableContent`). The `beginContentAccess` (page 1562) method marks the object's bytes as "accessed," thus protecting them from being discarded, and must be called before accessing the object, or else an exception will be raised. This method returns `YES` if the bytes have not been discarded and if they have been successfully marked as "accessed". Any method that directly or indirectly accesses these bytes or their length when they are not "accessed" will raise an exception. When you are done with the data, call `endContentAccess` (page 1563) to allow them to be discarded in order to quickly free up memory.

You may use these objects by themselves, and do not necessarily have to use them in conjunction with `NSCache` to get the purging behavior. The `NSCache` class incorporates a caching mechanism with some auto-removal policies to ensure that its memory footprint does not get too large.

`NSPurgeableData` objects should not be used as keys in hashing-based collections, because the value of the bytes pointer can change after every mutation of the data.

Adopted Protocols

NSDiscardableContent

```
beginContentAccess
endContentAccess
discardContentIfPossible
isContentDiscarded
```

NSRecursiveLock Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

`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 1609)
- `unlock` (page 1610)

Tasks

Acquiring a Lock

`- lockBeforeDate:` (page 1076)
Attempts to acquire a lock before a given date.

`- tryLock` (page 1077)
Attempts to acquire a lock, and immediately returns a Boolean value that indicates whether the attempt was successful.

Naming the Lock

- `setName:` (page 1077)
  Assigns a name to the receiver
- `name` (page 1076)
  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 iOS 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 iOS 2.0 and later.

See Also
- `setName:` (page 1077)

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 iOS 2.0 and later.

See Also

- name (page 1076)

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 iOS 2.0 and later.

Declared In

NSLock.h

NSRegularExpression Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSRegularExpression.h

Overview

The `NSRegularExpression` class is used to represent and apply regular expressions to Unicode strings. An instance of this class is an immutable representation of a compiled regular expression pattern and various option flags. The pattern syntax currently supported is that specified by ICU.

The fundamental matching method for `NSRegularExpression` is a Block iterator method that allows clients to supply a Block object which will be invoked each time the regular expression matches a portion of the target string. There are additional convenience methods for returning all the matches as an array, the total number of matches, the first match, and the range of the first match.

An individual match is represented by an instance of the `NSTextCheckingResult` class, which carries information about the overall matched range (via its `range` (page 1294) property), and the range of each individual capture group (via the `rangeAtIndex:` (page 1303) method). For basic `NSRegularExpression` objects, these match results will be of type `NSTextCheckingTypeRegularExpression` (page 1307), but subclasses may use other types.

Examples Using NSRegularExpression

What follows are a set of graduated examples for using the `NSRegularExpression` class. All these examples use the regular expression `\\b(a|b)(c|d)\\b` as their regular expression.

This snippet creates a regular expression to match two-letter words, in which the first letter is "a" or "b" and the second letter is "c" or "d". Specifying `NSRegularExpressionCaseInsensitive` (page 1099) means that matches will be case-insensitive, so this will match "BC", "aD", and so forth, as well as their lower-case equivalents.

```
NSError *error = NULL;
NSRegularExpression *regex = [NSRegularExpression
regularExpressionWithPattern:@"\\b(a|b)(c|d)\\b"
```

```
options:NSRegularExpressionCaseInsensitive

error:&error];
```

The `numberOfMatchesInString:options:range:` (page 1095) method provides a simple mechanism for counting the number of matches in a given range of a string.

```
NSUInteger numberOfMatches = [regex numberOfMatchesInString:string
                                                    options:0
                                                      range:NSMakeRange(0,
[string length])];
```

If you are interested only in the overall range of the first match, the `rangeOfFirstMatchInString:options:range:` (page 1096) method provides it for you. Some regular expressions (though not the example pattern) can successfully match a zero-length range, so the comparison of the resulting range with `{NSNotFound, 0}` is the most reliable way to determine whether there was a match or not.

The example regular expression contains two capture groups, corresponding to the two sets of parentheses, one for the first letter, and one for the second. If you are interested in more than just the overall matched range, you want to obtain an `NSTextCheckingResult` object corresponding to a given match. This object provides information about the overall matched range, via its `range` (page 1294) property, and also supplies the capture group ranges, via the `rangeAtIndex:` (page 1303) method. The first capture group range is given by `[result rangeAtIndex:1]`, the second by `[result rangeAtIndex:2]`. Sending a result the `rangeAtIndex:` (page 1303) message and passing `0` is equivalent to `[result range]`.

If the result returned is non-`nil`, then `[result range]` will always be a valid range, so it is not necessary to compare it against `{NSNotFound, 0}`. However, for some regular expressions (though not the example pattern) some capture groups may or may not participate in a given match. If a given capture group does not participate in a given match, then `[result rangeAtIndex:idx]` will return `{NSNotFound, 0}`.

```
NSRange rangeOfFirstMatch = [regex rangeOfFirstMatchInString:string options:0
range:NSMakeRange(0, [string length])];
if (!NSEqualRanges(rangeOfFirstMatch, NSMakeRange(NSNotFound, 0))) {
    NSString *substringForFirstMatch = [string
substringWithRange:rangeOfFirstMatch];
}
```

The `firstMatchInString:options:range:` (page 1093) returns only the first match.

```
NSArray *matches = [regex matchesInString:string
                                  options:0
                                    range:NSMakeRange(0, [string length])];
for (NSTextCheckingResult *match in matches) {
     NSRange matchRange = [match range];
     NSRange firstHalfRange = [match rangeAtIndex:1];
     NSRange secondHalfRange = [match rangeAtIndex:2];
}
```

The `matchesInString:options:range:` (page 1095) method is similar to `firstMatchInString:options:range:` (page 1093) but it returns all the matching results.

```
NSTextCheckingResult *match = [regex firstMatchInString:string
                                                 options:0
                                                   range:NSMakeRange(0, [string
 length])];
```

```
if (match) {
    NSRange matchRange = [match range];
    NSRange firstHalfRange = [match rangeAtIndex:1];
    NSRange secondHalfRange = [match rangeAtIndex:2];
 }
}
```

The Block enumeration method `enumerateMatchesInString:options:range:usingBlock:` (page 1091) is the most general and flexible of the matching methods of `NSRegularExpression`. It allows you to iterate through matches in a string, performing arbitrary actions on each as specified by the code in the Block and to stop partway through if desired. In the following example case, the iteration is stopped after a certain number of matches have been found.

If neither of the special options `NSMatchingReportProgress` (page 1101) or `NSMatchingReportCompletion` (page 1101) is specified, then the result argument to the Block is guaranteed to be non-`nil`, and as mentioned before, it is guaranteed to have a valid overall range. See "NSMatchingOptions" (page 1100) for the significance of `NSMatchingReportProgress` (page 1101) or `NSMatchingReportCompletion` (page 1101).

```
__block NSUInteger count = 0;
[regex enumerateMatchesInString:string options:0 range:NSMakeRange(0, [string
length]) usingBlock:^(NSTextCheckingResult *match, NSMatchingFlags flags, BOOL
 *stop){
     NSRange matchRange = [match range];
     NSRange firstHalfRange = [match rangeAtIndex:1];
     NSRange secondHalfRange = [match rangeAtIndex:2];
     if (++count >= 100) *stop = YES;
}];
```

`NSRegularExpression` also provides simple methods for performing find-and-replace operations on a string. The following example returns a modified copy, but there is a corresponding method for modifying a mutable string in place. The template specifies what is to be used to replace each match, with `$0` representing the contents of the overall matched range, `$1` representing the contents of the first capture group, and so on. In this case, the template reverses the two letters of the word.

```
NSString *modifiedString = [regex stringByReplacingMatchesInString:string
                                                            options:0
                                                           range:NSMakeRange(0,
 [string length])
                                                      withTemplate:@"$2$1"];
```

Concurrency and Thread Safety

`NSRegularExpression` is designed to be immutable and thread safe, so that a single instance can be used in matching operations on multiple threads at once. However, the string on which it is operating should not be mutated during the course of a matching operation, whether from another thread or from within the Block used in the iteration.

Regular Expression Syntax

The following tables describe the character expressions used by the regular expression to match patterns within a string, the pattern operators that specify how many times a pattern is matched and additional matching restrictions, and the last table specifies flags that can be included in the regular expression pattern that specify search behavior over multiple lines (these flags can also be specified using the "NSRegularExpressionOptions" (page 1099) option flags.

Regular Expression Metacharacters

Table 79-1 (page 1082) describe the character sequences used to match characters within a string.

Table 79-1 Regular Expression Metacharacters

Character Expression	Description
`\a`	Match a BELL, `\u0007`
`\A`	Match at the beginning of the input. Differs from `^` in that `\A` will not match after a new line within the input.
`\b, outside of a [Set]`	Match if the current position is a word boundary. Boundaries occur at the transitions between word (`\w`) and non-word (`\W`) characters, with combining marks ignored. For better word boundaries, see `NSRegularExpressionUseUnicodeWordBoundaries` (page 1100).
`\b, within a [Set]`	Match a BACKSPACE, `\u0008`.
`\B`	Match if the current position is not a word boundary.
`\cX`	Match a `control-X` character
`\d`	Match any character with the Unicode General Category of Nd (Number, Decimal Digit.)
`\D`	Match any character that is not a decimal digit.
`\e`	Match an `ESCAPE`, `\u001B`.
`\E`	Terminates a `\Q ... \E` quoted sequence.
`\f`	Match a FORM FEED, `\u000C`.
`\G`	Match if the current position is at the end of the previous match.
`\n`	Match a `LINE FEED`, `\u000A`.
`\N{`*UNICODE CHARACTER NAME*`}`	Match the named character.
`\p{`*UNICODE PROPERTY NAME*`}`	Match any character with the specified Unicode Property.

Character Expression	Description
\P{*UNICODE PROPERTY NAME*}	Match any character not having the specified Unicode Property.
\Q	Quotes all following characters until \E.
\r	Match a CARRIAGE RETURN, \u000D.
\s	Match a white space character. White space is defined as [\t\n\f\r\p{Z}].
\S	Match a non-white space character.
\t	Match a HORIZONTAL TABULATION, \u0009.
\u*hhhh*	Match the character with the hex value *hhhh*.
\U*hhhhhhhh*	Match the character with the hex value *hhhhhhhh*. Exactly eight hex digits must be provided, even though the largest Unicode code point is \U0010ffff.
\w	Match a word character. Word characters are [\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}].
\W	Match a non-word character.
\x{*hhhh*}	Match the character with hex value *hhhh*. From one to six hex digits may be supplied.
\x*hh*	Match the character with two digit hex value *hh*.
\X	Match a Grapheme Cluster.
\Z	Match if the current position is at the end of input, but before the final line terminator, if one exists.
\z	Match if the current position is at the end of input.
n	Back Reference. Match whatever the *n*th capturing group matched. *n* must be a number > 1 and < total number of capture groups in the pattern.
\0*ooo*	Match an Octal character. *ooo* is from one to three octal digits. 0377 is the largest allowed Octal character. The leading zero is required; it distinguishes Octal constants from back references.
[*pattern*]	Match any one character from the pattern.
.	Match any character. See also NSRegularExpressionDotMatchesLineSeparators (page 1099) and the s character expression above.
^	Match at the beginning of a line. See also NSRegularExpressionAnchorsMatchLines (page 1099) and the m character expression above.

Character Expression	Description
$	Match at the end of a line. See also NSRegularExpressionAnchorsMatchLines (page 1099) and the m character expression above.
\	Quotes the following character. Characters that must be quoted to be treated as literals are * ? + [( ) { } ^ $ | \ . /

Regular Expression Operators

Table 79-2 defines the regular expression operators.

Table 79-2 Regular Expression Operators

Operator	Description
|	Alternation. *A* | *B* matches either *A* or *B*.
*	Match 0 or more times. Match as many times as possible.
+	Match 1 or more times. Match as many times as possible.
?	Match zero or one times. Prefer one.
{*n*}	Match exactly *n* times.
{*n*,}	Match at least *n* times. Match as many times as possible.
{*n*,*m*}	Match between *n* and *m* times. Match as many times as possible, but not more than *m*.
*?	Match 0 or more times. Match as few times as possible.
+?	Match 1 or more times. Match as few times as possible.
??	Match zero or one times. Prefer zero.
{*n*}?	Match exactly n times.
{*n*,}?	Match at least n times, but no more than required for an overall pattern match.
{*n*,*m*}?	Match between n and m times. Match as few times as possible, but not less than n.
*+	Match 0 or more times. Match as many times as possible when first encountered, do not retry with fewer even if overall match fails (Possessive Match).
++	Match 1 or more times. Possessive match.
?+	Match zero or one times. Possessive match.
{*n*}+	Match exactly *n* times.

Operator	Description
`{n,}+`	Match at least *n* times. Possessive Match.
`{n,m}+`	Match between *n* and *m* times. Possessive Match.
`(...)`	Capturing parentheses. Range of input that matched the parenthesized subexpression is available after the match.
`(?:...)`	Non-capturing parentheses. Groups the included pattern, but does not provide capturing of matching text. Somewhat more efficient than capturing parentheses.
`(?>...)`	Atomic-match parentheses. First match of the parenthesized subexpression is the only one tried; if it does not lead to an overall pattern match, back up the search for a match to a position before the "`(?>`"
`(?# ... )`	Free-format comment `(?# comment )`.
`(?= ... )`	Look-ahead assertion. True if the parenthesized pattern matches at the current input position, but does not advance the input position.
`(?! ... )`	Negative look-ahead assertion. True if the parenthesized pattern does not match at the current input position. Does not advance the input position.
(?<= ...)	Look-behind assertion. True if the parenthesized pattern matches text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators.)
`(?<! ... )`	Negative Look-behind assertion. True if the parenthesized pattern does not match text preceding the current input position, with the last character of the match being the input character just before the current position. Does not alter the input position. The length of possible strings matched by the look-behind pattern must not be unbounded (no * or + operators.)
`(?ismwx-ismwx: ... )`	Flag settings. Evaluate the parenthesized expression with the specified flags enabled or -disabled. The flags are defined in “Flag Options.”
`(?ismwx-ismwx)`	Flag settings. Change the flag settings. Changes apply to the portion of the pattern following the setting. For example, (?i) changes to a case insensitive match.The flags are defined in “Flag Options.”

Template Matching Format

The `NSRegularExpression` class provides find-and-replace methods for both immutable and mutable strings using the technique of template matching. Table 79-3 (page 1086) describes the syntax.

Table 79-3 Template Matching Format

Character	Descriptions
`$`*n*	The text of capture group n will be substituted for $*n*. *n* must be `>= 0` and not greater than the number of capture groups. A `$` not followed by a digit has no special meaning, and will appear in the substitution text as itself, a `$`.
`\`	Treat the following character as a literal, suppressing any special meaning. Backslash escaping in substitution text is only required for '$' and '\', but may be used on any other character without bad effects.

The replacement string is treated as a template, with `$0` being replaced by the contents of the matched range, `$1` by the contents of the first capture group, and so on. Additional digits beyond the maximum required to represent the number of capture groups will be treated as ordinary characters, as will a `$` not followed by digits. Backslash will escape both `$` and `\`.

Flag Options

The following flags control various aspects of regular expression matching. These flag values may be specified within the pattern using the `(?ismx-ismx)` pattern options. Equivalent behaviors can be specified for the entire pattern when an `NSRegularExpression` is initialized, using the "NSRegularExpressionOptions" (page 1099) option flags.

Table 79-4 Flag Options

Flag (Pattern)	Description
i	If set, matching will take place in a case-insensitive manner.
x	If set, allow use of white space and #comments within patterns
s	If set, a "." in a pattern will match a line terminator in the input text. By default, it will not. Note that a `carriage-return / line-feed pair` in text behave as a single line terminator, and will match a single "." in a regular expression pattern
m	Control the behavior of "^" and "$" in a pattern. By default these will only match at the start and end, respectively, of the input text. If this flag is set, "^" and "$" will also match at the start and end of each line within the input text.
Para	Controls the behavior of `\b` in a pattern. If set, word boundaries are found according to the definitions of word found in Unicode UAX 29, Text Boundaries. By default, word boundaries are identified by means of a simple classification of characters as either "word" or "non-word", which approximates traditional regular expression behavior. The results obtained with the two options can be quite different in runs of spaces and other non-word characters.

ICU License

Table 79-1 (page 1082), Table 79-2 (page 1084), Table 79-3 (page 1086), Table 79-4 (page 1086) are reproduced from the ICU User Guide, Copyright (c) 2000 - 2009 IBM and Others, which are licensed under the following terms:

COPYRIGHT AND PERMISSION NOTICE

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Tasks

Creating Regular Expressions

`+ regularExpressionWithPattern:options:error:` (page 1091)
Creates an `NSRegularExpression` instance with the specified regular expression pattern and options.

`- initWithPattern:options:error:` (page 1094)
Returns an initialized `NSRegularExpression` instance with the specified regular expression pattern and options.

Getting the Regular Expression and Options

`pattern` (page 1089) *property*
Returns the regular expression pattern. (read-only)

`options` (page 1089) *property*
Returns the options used when the regular expression option was created. (read-only)

`numberOfCaptureGroups` (page 1089) *property*
Returns the number of capture groups in the regular expression. (read-only)

Searching Strings Using Regular Expressions

- `numberOfMatchesInString:options:range:` (page 1095)
  Returns the number of matches of the regular expression within the specified range of the string.
- `enumerateMatchesInString:options:range:usingBlock:` (page 1091)
  Enumerates the string allowing the Block to handle each regular expression match.
- `matchesInString:options:range:` (page 1095)
  Returns an array containing all the matches of the regular expression in the string.
- `firstMatchInString:options:range:` (page 1093)
  Returns the first match of the regular expression within the specified range of the string.
- `rangeOfFirstMatchInString:options:range:` (page 1096)
  Returns the range of the first match of the regular expression within the specified range of the string.

Replacing Strings Using Regular Expressions

- `replaceMatchesInString:options:range:withTemplate:` (page 1097)
  Replaces regular expression matches within the mutable string the using the template string.
- `stringByReplacingMatchesInString:options:range:withTemplate:` (page 1098)
  Returns a new string containing matching regular expressions replaced with the template string.

Escaping Characters in a String

+ `escapedTemplateForString:` (page 1090)
  Returns a template string by adding backslash escapes as necessary to protect any characters that would match as pattern metacharacters.
+ `escapedPatternForString:` (page 1090)
  Returns a string by adding backslash escapes as necessary to protect any characters that would match as pattern metacharacters.

Custom Replace Functionality

- `replacementStringForResult:inString:offset:template:` (page 1097)
  Used to perform template substitution for a single result for clients implementing their own replace functionality.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

numberOfCaptureGroups

Returns the number of capture groups in the regular expression. (read-only)

```
@property(readonly) NSUInteger numberOfCaptureGroups
```

Discussion
A capture group consists of each possible match within a regular expression. Each capture group can then be used in a replacement template to insert that value into a replacement string.

This value puts a limit on the values of `n` for `$n` in templates, and it determines the number of ranges in the returned `NSTextCheckingResult` instances returned in the `match...` methods.

An exception will be generated if you attempt to access a result with an index value exceeding `numberOfCaptureGroups-1`.

Availability
Available in iOS 4.0 and later.

Declared In
`NSRegularExpression.h`

options

Returns the options used when the regular expression option was created. (read-only)

```
@property(readonly) NSRegularExpressionOptions options
```

Discussion
The options property specifies aspects of the regular expression matching that are always used when matching the regular expression. For example, if the expression is case sensitive, allows comments, ignores metacharacters, etc.. See "NSRegularExpressionOptions" (page 1099) for a complete discussion of the possible constants and their meanings.

Availability
Available in iOS 4.0 and later.

See Also
`+ regularExpressionWithPattern:options:error:` (page 1091)
`- initWithPattern:options:error:` (page 1094)

Declared In
`NSRegularExpression.h`

pattern

Returns the regular expression pattern. (read-only)

```
@property(readonly) NSString *pattern
```

Availability
Available in iOS 4.0 and later.

See Also
`+ regularExpressionWithPattern:options:error:` (page 1091)
`- initWithPattern:options:error:` (page 1094)

Declared In
`NSRegularExpression.h`

Class Methods

escapedPatternForString:

Returns a string by adding backslash escapes as necessary to protect any characters that would match as pattern metacharacters.

```
+ (NSString *)escapedPatternForString:(NSString *)string
```

Parameters
string
The string.

Return Value
The escaped string.

Discussion
Returns a string by adding backslash escapes as necessary to the given string, to escape any characters that would otherwise be treated as pattern metacharacters.

See "Flag Options" (page 1086) for the format of *template*.

Availability
Available in iOS 4.0 and later.

Declared In
`NSRegularExpression.h`

escapedTemplateForString:

Returns a template string by adding backslash escapes as necessary to protect any characters that would match as pattern metacharacters.

```
+ (NSString *)escapedTemplateForString:(NSString *)string
```

Parameters
string
The template string.

Return Value
The escaped template string.

Discussion
Returns a string by adding backslash escapes as necessary to the given string, to escape any characters that would otherwise be treated as pattern metacharacters.

See "Flag Options" (page 1086) for the format of *template*.

Availability
Available in iOS 4.0 and later.

Declared In
NSRegularExpression.h

regularExpressionWithPattern:options:error:

Creates an NSRegularExpression instance with the specified regular expression pattern and options.

```
+ (NSRegularExpression *)regularExpressionWithPattern:(NSString *)pattern
    options:(NSRegularExpressionOptions)options error:(NSError **)error
```

Parameters

pattern
The regular expression pattern to compile.

options
The matching options. See "NSRegularExpressionOptions" (page 1099) for possible values. The values can be combined using the C-bitwise OR operator.

error
An out value that returns any error encountered during initialization. Returns nil if the regular expression pattern is invalid.

Return Value
An instance of NSRegularExpression for the specified regular expression and options.

Availability
Available in iOS 4.0 and later.

See Also
- initWithPattern:options:error: (page 1094)

Declared In
NSRegularExpression.h

Instance Methods

enumerateMatchesInString:options:range:usingBlock:

Enumerates the string allowing the Block to handle each regular expression match.

```
- (void)enumerateMatchesInString:(NSString *)string
    options:(NSMatchingOptions)options range:(NSRange)range usingBlock:(void
    (^)(NSTextCheckingResult *result, NSMatchingFlags flags, BOOL *stop))block
```

Parameters

string
The string.

options

The matching options to report. See "NSMatchingOptions" (page 1100) for the supported values.

range

The range of the string to test.

block

The Block enumerates the matches of the regular expression in the string..

The block takes three arguments:

result

An `NSTextCheckingResult` specifying the match. This result gives the overall matched range via its `range` (page 1294) property, and the range of each individual capture group via its `rangeAtIndex:` (page 1303) method. The range {`NSNotFound`, 0} is returned if one of the capture groups did not participate in this particular match.

flags

The current state of the matching progress. See "NSMatchingFlags" (page 1100) for the possible values.

stop

A reference to a Boolean value. The Block can set the value to `YES` to stop further processing of the array. The stop argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

The Block returns void.

Discussion

This method is the fundamental matching method for regular expressions and is suitable for overriding by subclassers. There are additional convenience methods for returning all the matches as an array, the total number of matches, the first match, and the range of the first match.

By default, the Block iterator method calls the Block precisely once for each match, with a non-`nil` `result` and the appropriate `flags`. The client may then stop the operation by setting the contents of `stop` to `YES`. The `stop` argument is an out-only argument. You should only ever set this Boolean to `YES` within the Block.

If the `NSMatchingReportProgress` (page 1101) matching option is specified, the Block will also be called periodically during long-running match operations, with `nil` result and `NSMatchingProgress` (page 1100) matching flag set in the Block's `flags` parameter, at which point the client may again stop the operation by setting the contents of stop to `YES`.

If the `NSMatchingReportCompletion` (page 1101) matching option is specified, the Block object will be called once after matching is complete, with `nil` result and the `NSMatchingCompleted` (page 1100) matching flag is set in the `flags` passed to the Block, plus any additional relevant "NSMatchingFlags" (page 1100) from among `NSMatchingHitEnd` (page 1100), `NSMatchingRequiredEnd` (page 1100), or `NSMatchingInternalError` (page 1100).

`NSMatchingProgress` (page 1100) and `NSMatchingCompleted` (page 1100) matching flags have no effect for methods other than this method.

The `NSMatchingHitEnd` (page 1100) matching flag is set in the `flags` passed to the Block if the current match operation reached the end of the search range. The `NSMatchingRequiredEnd` (page 1100) matching flag is set in the `flags` passed to the Block if the current match depended on the location of the end of the search range.

The "NSMatchingFlags" (page 1100) matching flag is set in the `flags` passed to the block if matching failed due to an internal error (such as an expression requiring exponential memory allocations) without examining the entire search range.

The `NSMatchingAnchored` (page 1101), `NSMatchingWithTransparentBounds` (page 1101), and `NSMatchingWithoutAnchoringBounds` (page 1101) regular expression options, specified in the `options` (page 1089) property specified when the regular expression instance is created, can apply to any match or replace method.

If `NSMatchingAnchored` (page 1101) matching option is specified, matches are limited to those at the start of the search range.

If `NSMatchingWithTransparentBounds` (page 1101) matching option is specified, matching may examine parts of the string beyond the bounds of the search range, for purposes such as word boundary detection, lookahead, etc.

If `NSMatchingWithoutAnchoringBounds` (page 1101) matching option is specified, `^` and `$` will not automatically match the beginning and end of the search range, but will still match the beginning and end of the entire string.

`NSMatchingWithTransparentBounds` (page 1101) and `NSMatchingWithoutAnchoringBounds` (page 1101) matching options have no effect if the search range covers the entire string.

Availability
Available in iOS 4.0 and later.

See Also
- `matchesInString:options:range:` (page 1095)
- `numberOfMatchesInString:options:range:` (page 1095)
- `firstMatchInString:options:range:` (page 1093)
- `rangeOfFirstMatchInString:options:range:` (page 1096)

Declared In
`NSRegularExpression.h`

firstMatchInString:options:range:

Returns the first match of the regular expression within the specified range of the string.

```
- (NSTextCheckingResult *)firstMatchInString:(NSString *)string
    options:(NSMatchingOptions)options range:(NSRange)range
```

Parameters

string
The string to search.

options
The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range
The range of the string to search.

Return Value
An `NSTextCheckingResult` object. This result gives the overall matched range via its `range` (page 1294) property, and the range of each individual capture group via its `rangeAtIndex:` (page 1303) method. The range {`NSNotFound`, 0} is returned if one of the capture groups did not participate in this particular match.

Discussion
This is a convenience method that calls `enumerateMatchesInString:options:range:usingBlock:` (page 1091).

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateMatchesInString:options:range:usingBlock:` (page 1091)
- `matchesInString:options:range:` (page 1095)
- `numberOfMatchesInString:options:range:` (page 1095)
- `rangeOfFirstMatchInString:options:range:` (page 1096)

Declared In
`NSRegularExpression.h`

initWithPattern:options:error:

Returns an initialized `NSRegularExpression` instance with the specified regular expression pattern and options.

```
- (id)initWithPattern:(NSString *)pattern options:(NSRegularExpressionOptions)options
    error:(NSError **)error
```

Parameters

pattern
The regular expression pattern to compile.

options
The regular expression options that are applied to the expression during matching. See "NSRegularExpressionOptions" (page 1099) for possible values.

error
An out value that returns any error encountered during initialization. Returns `nil` if the regular expression pattern is invalid.

Return Value
An instance of `NSRegularExpression` for the specified regular expression and options.

Availability
Available in iOS 4.0 and later.

See Also
- `regularExpressionWithPattern:options:error:` (page 1091)

Declared In
`NSRegularExpression.h`

matchesInString:options:range:

Returns an array containing all the matches of the regular expression in the string.

```
- (NSArray *)matchesInString:(NSString *)string options:(NSMatchingOptions)options
    range:(NSRange)range
```

Parameters

string

The string to search.

options

The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range

The range of the string to search.

Return Value

An array of `NSTextCheckingResult` objects. Each result gives the overall matched range via its `range` (page 1294) property, and the range of each individual capture group via its `rangeAtIndex:` (page 1303) method. The range {`NSNotFound`, 0} is returned if one of the capture groups did not participate in this particular match.

Discussion

This is a convenience method that calls `enumerateMatchesInString:options:range:usingBlock:` (page 1091) passing the appropriate string, options, and range..

Availability

Available in iOS 4.0 and later.

See Also

- `enumerateMatchesInString:options:range:usingBlock:` (page 1091)
- `numberOfMatchesInString:options:range:` (page 1095)
- `firstMatchInString:options:range:` (page 1093)
- `rangeOfFirstMatchInString:options:range:` (page 1096)

Declared In

`NSRegularExpression.h`

numberOfMatchesInString:options:range:

Returns the number of matches of the regular expression within the specified range of the string.

```
- (NSUInteger)numberOfMatchesInString:(NSString *)string
    options:(NSMatchingOptions)options range:(NSRange)range
```

Parameters

string

The string to search.

options

The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range

The range of the string to search.

Return Value
The number of matches of the regular expression.

Discussion
This is a convenience method that calls `enumerateMatchesInString:options:range:usingBlock:` (page 1091).

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateMatchesInString:options:range:usingBlock:` (page 1091)
- `matchesInString:options:range:` (page 1095)
- `firstMatchInString:options:range:` (page 1093)
- `rangeOfFirstMatchInString:options:range:` (page 1096)

Declared In
`NSRegularExpression.h`

rangeOfFirstMatchInString:options:range:

Returns the range of the first match of the regular expression within the specified range of the string.

```
- (NSRange)rangeOfFirstMatchInString:(NSString *)string
    options:(NSMatchingOptions)options range:(NSRange)range
```

Parameters

string
The string to search.

options
The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range
The range of the string to search.

Return Value
The range of the first match. Returns `{NSNotFound, 0}` if no match is found.

Discussion
This is a convenience method that calls `enumerateMatchesInString:options:range:usingBlock:` (page 1091).

Availability
Available in iOS 4.0 and later.

See Also
- `enumerateMatchesInString:options:range:usingBlock:` (page 1091)
- `matchesInString:options:range:` (page 1095)
- `numberOfMatchesInString:options:range:` (page 1095)
- `firstMatchInString:options:range:` (page 1093)

Declared In
`NSRegularExpression.h`

replaceMatchesInString:options:range:withTemplate:

Replaces regular expression matches within the mutable string the using the template string.

```
- (NSUInteger)replaceMatchesInString:(NSMutableString *)string
    options:(NSMatchingOptions)options range:(NSRange)range withTemplate:(NSString
     *)template
```

Parameters

string
The mutable string to search and replace values within.

options
The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range
The range of the string to search.

template
The substitution template used when replacing matching instances.

Return Value
The number of matches.

Discussion
See "Flag Options" (page 1086) for the format of *template*.

Availability
Available in iOS 4.0 and later.

See Also
- stringByReplacingMatchesInString:options:range:withTemplate: (page 1098)

Declared In
NSRegularExpression.h

replacementStringForResult:inString:offset:template:

Used to perform template substitution for a single result for clients implementing their own replace functionality.

```
- (NSString *)replacementStringForResult:(NSTextCheckingResult *)result
    inString:(NSString *)string offset:(NSInteger)offset template:(NSString
    *)template
```

Parameters

result
The result of the single match.

string
The string from which the result was matched.

offset
The offset to be added to the location of the result in the string.

template
See "Flag Options" (page 1086) for the format of *template*.

Return Value
A replacement string.

Discussion
For clients implementing their own replace functionality, this is a method to perform the template substitution for a single result, given the string from which the result was matched, an offset to be added to the location of the result in the string (for example, in cases that modifications to the string moved the result since it was matched), and a replacement template.

This is an advanced method that is used only if you wanted to iterate through a list of matches yourself and do the template replacement for each one, plus maybe some other calculation that you want to do in code, then you would use this at each step.

Availability
Available in iOS 4.0 and later.

Declared In
NSRegularExpression.h

stringByReplacingMatchesInString:options:range:withTemplate:

Returns a new string containing matching regular expressions replaced with the template string.

```
- (NSString *)stringByReplacingMatchesInString:(NSString *)string
    options:(NSMatchingOptions)options range:(NSRange)range withTemplate:(NSString
     *)template
```

Parameters

string
The string to search for values within.

options
The matching options to use. See "NSMatchingOptions" (page 1100) for possible values.

range
The range of the string to search.

template
The substitution template used when replacing matching instances.

Return Value
A string with matching regular expressions replaced by the template string.

Discussion
See "Flag Options" (page 1086) for the format of *template*.

Availability
Available in iOS 4.0 and later.

See Also
- replaceMatchesInString:options:range:withTemplate: (page 1097)

Declared In
NSRegularExpression.h

Constants

NSRegularExpressionOptions

These constants define the regular expression options. These constants are used by the property `options` (page 1089), `regularExpressionWithPattern:options:error:` (page 1091), and `initWithPattern:options:error:` (page 1094).

```
enum {
   NSRegularExpressionCaseInsensitive             = 1 << 0,
   NSRegularExpressionAllowCommentsAndWhitespace  = 1 << 1,
   NSRegularExpressionIgnoreMetacharacters        = 1 << 2,
   NSRegularExpressionDotMatchesLineSeparators    = 1 << 3,
   NSRegularExpressionAnchorsMatchLines           = 1 << 4,
   NSRegularExpressionUseUnixLineSeparators       = 1 << 5,
   NSRegularExpressionUseUnicodeWordBoundaries    = 1 << 6
};
typedef NSUInteger NSRegularExpressionOptions;
```

Constants

`NSRegularExpressionCaseInsensitive`
Match letters in the pattern independent of case.
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionAllowCommentsAndWhitespace`
Ignore whitespace and #-prefixed comments in the pattern.
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionIgnoreMetacharacters`
Treat the entire pattern as a literal string.
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionDotMatchesLineSeparators`
Allow `.` to match any character, including line separators.
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionAnchorsMatchLines`
Allow `^` and `$` to match the start and end of lines.
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionUseUnixLineSeparators`
Treat only `\n` as a line separator (otherwise, all standard line separators are used).
Available in iOS 4.0 and later.
Declared in `NSRegularExpression.h`.

`NSRegularExpressionUseUnicodeWordBoundaries`
Use Unicode `TR#29` to specify word boundaries (otherwise, traditional regular expression word boundaries are used).

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

NSMatchingFlags

Set by the Block as the matching progresses, completes, or fails. Used by the method `enumerateMatchesInString:options:range:usingBlock:` (page 1091).

```
enum {
   NSMatchingProgress               = 1 << 0,
   NSMatchingCompleted              = 1 << 1,
   NSMatchingHitEnd                 = 1 << 2,
   NSMatchingRequiredEnd            = 1 << 3,
   NSMatchingInternalError          = 1 << 4
};
typedef NSUInteger NSMatchingFlags;
```

Constants

`NSMatchingProgress`
Set when the Block is called to report progress during a long-running match operation.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingCompleted`
Set when the Block is called after matching has completed.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingHitEnd`
Set when the current match operation reached the end of the search range.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingRequiredEnd`
Set when the current match depended on the location of the end of the search range.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingInternalError`
Set when matching failed due to an internal error.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

NSMatchingOptions

The matching options constants specify the reporting, completion and matching rules to the expression matching methods. These constants are used by all methods that search for, or replace values, using a regular expression.

```
enum {
   NSMatchingReportProgress         = 1 << 0,
   NSMatchingReportCompletion       = 1 << 1,
   NSMatchingAnchored               = 1 << 2,
   NSMatchingWithTransparentBounds  = 1 << 3,
   NSMatchingWithoutAnchoringBounds = 1 << 4
};
typedef NSUInteger NSMatchingOptions;
```

Constants

`NSMatchingReportProgress`

Call the Block periodically during long-running match operations. This option has no effect for methods other than `enumerateMatchesInString:options:range:usingBlock:` (page 1091). See `enumerateMatchesInString:options:range:usingBlock:` (page 1091) for a description of the constant in context.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingReportCompletion`

Call the Block once after the completion of any matching. This option has no effect for methods other than `enumerateMatchesInString:options:range:usingBlock:` (page 1091). See `enumerateMatchesInString:options:range:usingBlock:` (page 1091) for a description of the constant in context.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingAnchored`

Specifies that matches are limited to those at the start of the search range. See `enumerateMatchesInString:options:range:usingBlock:` (page 1091) for a description of the constant in context.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingWithTransparentBounds`

Specifies that matching may examine parts of the string beyond the bounds of the search range, for purposes such as word boundary detection, lookahead, etc. This constant has no effect if the search range contains the entire string. See `enumerateMatchesInString:options:range:usingBlock:` (page 1091) for a description of the constant in context.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

`NSMatchingWithoutAnchoringBounds`

Specifies that `^` and `$` will not automatically match the beginning and end of the search range, but will still match the beginning and end of the entire string. This constant has no effect if the search range contains the entire string. See `enumerateMatchesInString:options:range:usingBlock:` (page 1091) for a description of the constant in context.

Available in iOS 4.0 and later.

Declared in `NSRegularExpression.h`.

NSRunLoop Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSRunLoop.h
Companion guide	Threading Programming Guide
Related sample code	CryptoExercise GLSprite WiTap

Overview

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.

Your application cannot 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 1105).

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 1105)
  Returns the `NSRunLoop` object for the current thread.

- `currentMode` (page 1108)
  Returns the receiver's current input mode.

- `limitDateForMode:` (page 1109)
  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 1105)
  Returns the run loop of the main thread.

- `getCFRunLoop` (page 1109)
  Returns the receiver's underlying *CFRunLoop Reference* object.

Managing Timers

- `addTimer:forMode:` (page 1107)
  Registers a given timer with a given input mode.

Managing Ports

- `addPort:forMode:` (page 1106)
  Adds a port as an input source to the specified mode of the run loop.

- `removePort:forMode:` (page 1111)
  Removes a port from the specified input mode of the run loop.

Running a Loop

- `run` (page 1111)
  Puts the receiver into a permanent loop, during which time it processes data from all attached input sources.

- `runMode:beforeDate:` (page 1112)
  Runs the loop once, blocking for input in the specified mode until a given date.

- `runUntilDate:` (page 1112)
  Runs the loop until the specified date, during which time it processes data from all attached input sources.

- `acceptInputForMode:beforeDate:` (page 1106)
  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 1110)
  Schedules the sending of a message on the current run loop.
- `cancelPerformSelector:target:argument:` (page 1107)
  Cancels the sending of a previously scheduled message.
- `cancelPerformSelectorsWithTarget:` (page 1108)
  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 iOS 2.0 and later.

See Also
- `currentMode` (page 1108)

Related Sample Code
CryptoExercise
GLSprite
WiTap

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 iOS 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 1113).

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 iOS 2.0 and later.

See Also
- runMode:beforeDate: (page 1112)

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 1113).

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 iOS 2.0 and later.

See Also
- removePort:forMode: (page 1111)

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 1113).

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 1332) message to the timer.

Availability
Available in iOS 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 1110) 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 iOS 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 1107), 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 iOS 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 1106) and `runMode:beforeDate:` (page 1112).

Availability
Available in iOS 2.0 and later.

See Also
`+ currentRunLoop` (page 1105)
`- limitDateForMode:` (page 1109)
`- run` (page 1111)
`- runUntilDate:` (page 1112)

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 iOS 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 1113).

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 iOS 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 1113).

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 iOS 2.0 and later.

See Also
- cancelPerformSelector:target:argument: (page 1107)

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 1113).

Discussion

If you added the port to multiple input modes, you must remove it from each mode separately.

Availability

Available in iOS 2.0 and later.

See Also

- addPort:forMode: (page 1106)

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 1112). 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 iOS 2.0 and later.

See Also
- `runUntilDate:` (page 1112)

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 1113).

limitDate

The date until which to block.

Return Value
`YES` if the run loop ran and processed an input source or if the specified timeout value was reached; otherwise, `NO` if the run loop could not be started.

Discussion
If no input sources or timers are attached to the run loop, this method exits immediately and returns `NO`; 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 does not guarantee that the run loop will exit immediately. 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 iOS 2.0 and later.

See Also
- `run` (page 1111)
- `runUntilDate:` (page 1112)

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 1112) 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 iOS 2.0 and later.

See Also

- `run` (page 1111)

Declared In

`NSRunLoop.h`

Constants

Run Loop Modes

`NSRunLoop` defines the following run loop mode.

```
extern NSString* const NSDefaultRunLoopMode;
extern NSString* const NSRunLoopCommonModes;
```

Constants

`NSDefaultRunLoopMode`

The mode to deal with input sources other than `NSConnection` objects.

This is the most commonly used run-loop mode.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSRunLoop.h`.

Declared In

`Foundation/NSRunLoop.h`

Additional run loop modes are defined by `NSConnection` and `NSApplication`.

`NSConnectionReplyMode`	

`NSModalPanelRunLoopMode`	
`NSEventTrackingRunLoopMode`	

NSScanner Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSScanner.h Foundation/NSDecimalNumber.h
Companion guide	String Programming Guide

Overview

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 1130) and all of the methods listed under "Configuring a Scanner" (page 1116) 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 1129) 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 1274)`[scanner scanLocation]]`.

Adopted Protocols

NSCopying

- copyWithZone: (page 1554)

Tasks

Creating a Scanner

+ scannerWithString: (page 1118)
Returns an NSScanner object that scans a given string.

+ localizedScannerWithString: (page 1117)
Returns an NSScanner object that scans a given string according to the user's default locale.

- initWithString: (page 1119)
Returns an NSScanner object initialized to scan a given string.

Getting a Scanner's String

- string (page 1130)
Returns the string with which the receiver was created or initialized.

Configuring a Scanner

- setScanLocation: (page 1130)
Sets the location at which the next scan operation will begin to a given index.

- scanLocation (page 1126)
Returns the character position at which the receiver will begin its next scanning operation.

- setCaseSensitive: (page 1128)
Sets whether the receiver is case sensitive when scanning characters.

- caseSensitive (page 1118)
Returns a Boolean value that indicates whether the receiver distinguishes case in the characters it scans.

- setCharactersToBeSkipped: (page 1129)
Sets the set of characters to ignore when scanning for a value representation.

- charactersToBeSkipped (page 1119)
Returns a character set containing the characters the receiver ignores when looking for a scannable element.

- setLocale: (page 1129)
Sets the receiver's locale to a given locale.

- locale (page 1120)
Returns the receiver's locale.

Scanning a String

- `scanCharactersFromSet:intoString:` (page 1121)
  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 1127)
  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 1121)
  Scans for an `NSDecimal` value, returning a found value by reference.
- `scanDouble:` (page 1122)
  Scans for a `double` value, returning a found value by reference.
- `scanFloat:` (page 1122)
  Scans for a `float` value, returning a found value by reference.
- `scanHexDouble:` (page 1123)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanHexFloat:` (page 1123)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanHexInt:` (page 1124)
  Scans for an `unsigned` value from a hexadecimal representation, returning a found value by reference.
- `scanHexLongLong:` (page 1124)
  Scans for a `double` value from a hexadecimal representation, returning a found value by reference.
- `scanInteger:` (page 1125)
  Scans for an `NSInteger` value from a decimal representation, returning a found value by reference
- `scanInt:` (page 1124)
  Scans for an `int` value from a decimal representation, returning a found value by reference.
- `scanLongLong:` (page 1126)
  Scans for a `long long` value from a decimal representation, returning a found value by reference.
- `scanString:intoString:` (page 1126)
  Scans a given string, returning an equivalent string object by reference if a match is found.
- `scanUpToString:intoString:` (page 1128)
  Scans the string until a given string is encountered, accumulating characters into a string that's returned by reference.
- `isAtEnd` (page 1120)
  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 1119) with *aString*. The locale is set with `setLocale:` (page 1129).

Availability

Available in iOS 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 1119) with *aString*.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also
- `setCaseSensitive:` (page 1128)
- `setCharactersToBeSkipped:` (page 1129)

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 1124) 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 1124) 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 iOS 2.0 and later.

See Also
- `setCharactersToBeSkipped:` (page 1129)
- `whitespaceAndNewlineCharacterSet` (page 193) (`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 iOS 2.0 and later.

See Also
`+ localizedScannerWithString:` (page 1117)
`+ scannerWithString:` (page 1118)

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 iOS 2.0 and later.

See Also
`- charactersToBeSkipped` (page 1119)

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 iOS 2.0 and later.

See Also
`- setLocale:` (page 1129)

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 iOS 2.0 and later.

See Also
- scanUpToCharactersFromSet:intoString: (page 1127)

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 iOS 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 iOS 2.0 and later.

See Also

`doubleValue` (page 1218) (`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 iOS 2.0 and later.

See Also

`floatValue` (page 1221) (`NSString`)

Declared In
NSScanner.h

scanHexDouble:

Scans for a double value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexDouble:(double *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value
YES if the receiver finds a valid double-point representation, otherwise NO.

Discussion
This corresponds to %a or %A formatting. The hexadecimal double representation must be preceded by 0x or 0X.

Invoke this method with NULL as *result* to simply scan past a hexadecimal double representation.

Availability
Available in iOS 2.0 and later.

Declared In
NSScanner.h

scanHexFloat:

Scans for a double value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexFloat:(float *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value
YES if the receiver finds a valid float-point representation, otherwise NO.

Discussion
This corresponds to %a or %A formatting. The hexadecimal float representation must be preceded by 0x or 0X.

Invoke this method with NULL as *result* to simply scan past a hexadecimal float representation.

Availability
Available in iOS 2.0 and later.

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 iOS 2.0 and later.

Declared In

`NSScanner.h`

scanHexLongLong:

Scans for a `double` value from a hexadecimal representation, returning a found value by reference.

```
- (BOOL)scanHexLongLong:(unsigned long long *)result
```

Parameters

result

Upon return, contains the scanned value.

Return Value

`YES` if the receiver finds a valid double-point representation, otherwise `NO`.

Discussion

Invoke this method with `NULL` as *result* to simply scan past a hexadecimal long long representation.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

intValue (page 1243) (NSString)

- scanInteger: (page 1125)

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 iOS 2.0 and later.

See Also

integerValue (page 1243) (NSString)

- scanInt: (page 1124)

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 iOS 2.0 and later.

See Also
- setScanLocation: (page 1130)

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 LLONG_MAX or LLONG_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 iOS 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 iOS 2.0 and later.

See Also
- scanUpToString:intoString: (page 1128)

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 1119) 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 iOS 2.0 and later.

See Also
- scanCharactersFromSet:intoString: (page 1121)

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 1119) 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 stopString is the first string in the receiver, then the method returns NO and *stringValue* is not changed.

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 iOS 2.0 and later.

See Also
- scanString:intoString: (page 1126)

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 iOS 2.0 and later.

See Also
- `caseSensitive` (page 1118)
- `setCharactersToBeSkipped:` (page 1129)

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. Pass `nil` to not ignore any characters.

Discussion
For example, if a scanner ignores spaces and you send it a `scanInt:` (page 1124) 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 1124) 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 iOS 2.0 and later.

See Also
- `charactersToBeSkipped` (page 1119)

`whitespaceAndNewlineCharacterSet` (page 193) (`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 iOS 2.0 and later.

See Also

`- locale` (page 1120)

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 1126) or `scanCharactersFromSet:intoString:` (page 1121), which allow you to verify that the expected substring (or set of characters) is in fact present.

Availability

Available in iOS 2.0 and later.

See Also

`- scanLocation` (page 1126)

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 iOS 2.0 and later.

See Also
`- locale` (page 1120)

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 iOS 2.0 and later.
Declared in	Foundation/NSSet.h Foundation/NSSortDescriptor.h
Companion guide	Collections Programming Topics
Related sample code	aurioTouch GKTank MoviePlayer ScrollViewSuite WiTap

Overview

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 1631) and `isEqual:` (page 1632)—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 1137) 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 1141) returns an array containing the objects in a set. `anyObject` (page 1142) returns some object in the set. `count` (page 1143) returns the number of objects currently in the set. `member:` (page 1151) returns the object in the set that is equal to a specified object. Additionally, `intersectsSet:` (page 1149) tests for set intersection, `isEqualToSet:` (page 1149) tests for set equality, and `isSubsetOfSet:` (page 1150) tests for one set being a subset of another.

The `objectEnumerator` (page 1152) 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 1150) and `makeObjectsPerformSelector:withObject:` (page 1151) 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating a Set

+ `set` (page 1137)
  Creates and returns an empty set.

+ `setWithArray:` (page 1138)
  Creates and returns a set containing a uniqued collection of those objects contained in a given array.

+ `setWithObject:` (page 1138)
  Creates and returns a set that contains a single given object.

+ `setWithObjects:` (page 1139)
  Creates and returns a set containing the objects in a given argument list.

+ `setWithObjects:count:` (page 1139)
  Creates and returns a set containing a specified number of objects from a given C array of objects.

+ `setWithSet:` (page 1140)
  Creates and returns a set containing the objects from another set.

- `setByAddingObject:` (page 1154)
  Returns a new set formed by adding a given object to the collection defined by the receiver.

- `setByAddingObjectsFromSet:` (page 1155)
  Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

- `setByAddingObjectsFromArray:` (page 1154)
  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 1146)
  Initializes a newly allocated set with the objects that are contained in a given array.

- `initWithObjects:` (page 1146)
  Initializes a newly allocated set with members taken from the specified list of objects.

- `initWithObjects:count:` (page 1147)
  Initializes a newly allocated set with a specified number of objects from a given C array of objects.

- `initWithSet:` (page 1147)
  Initializes a newly allocated set and adds to it objects from another given set.

- `initWithSet:copyItems:` (page 1148)
  Initializes a newly allocated set and adds to it members of another given set.

Counting Entries

- `count` (page 1143)
  Returns the number of members in the receiver.

Accessing Set Members

- `allObjects` (page 1141)
  Returns an array containing the receiver's members, or an empty array if the receiver has no members.
- `anyObject` (page 1142)
  Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.
- `containsObject:` (page 1142)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `filteredSetUsingPredicate:` (page 1145)
  Evaluates a given predicate against each object in the receiver and returns a new set containing the objects for which the predicate returns true.
- `makeObjectsPerformSelector:` (page 1150)
  Sends to each object in the receiver a message specified by a given selector.
- `makeObjectsPerformSelector:withObject:` (page 1151)
  Sends to each object in the receiver a message specified by a given selector.
- `member:` (page 1151)
  Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.
- `objectEnumerator` (page 1152)
  Returns an enumerator object that lets you access each object in the receiver.
- `enumerateObjectsUsingBlock:` (page 1144)
  Executes a given Block using each object in the receiver.
- `enumerateObjectsWithOptions:usingBlock:` (page 1144)
  Executes a given Block using each object in the receiver, using the specified enumeration options.
- `objectsPassingTest:` (page 1152)
  Returns a set of object that pass a test in a given Block.
- `objectsWithOptions:passingTest:` (page 1153)
  Returns a set of object that pass a test in a given Block, using the specified enumeration options.

Comparing Sets

- `isSubsetOfSet:` (page 1150)
  Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.
- `intersectsSet:` (page 1149)
  Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.
- `isEqualToSet:` (page 1149)
  Compares the receiver to another set.
- `valueForKey:` (page 1156)
  Return a set containing the results of invoking `valueForKey:` on each of the receiver's members.
- `setValue:forKey:` (page 1155)
  Invokes `setValue:forKey:` on each of the receiver's members.

Creating a Sorted Array

- `sortedArrayUsingDescriptors:` (page 1156)
  Returns an array of the receiver's content sorted as specified by a given array of sort descriptors.

Key-Value Observing

- `addObserver:forKeyPath:options:context:` (page 1141)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 1153)
  Raises an exception.

Describing a Set

- `description` (page 1143)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:` (page 1143)
  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 iOS 2.0 and later.

See Also
`+ setWithArray:` (page 1138)
`+ setWithObject:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 in *anArray*, it is added only once to the returned set. Each object receives a `retain` (page 1638) 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 iOS 2.0 and later.

See Also

`+ set` (page 1137)
`+ setWithObject:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 1638) message after being added to the set.

Return Value

A new set that contains a single member, *anObject*.

Availability

Available in iOS 2.0 and later.

See Also

`+ set` (page 1137)
`+ setWithArray:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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.

anObject, ...
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 of objects, it is added only once to the returned set. Each object receives a `retain` (page 1638) 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 numberWithInteger:5];
NSString *aString = @"a string";

mySet = [NSSet setWithObjects:someData, aValue, aString, nil];
```

Availability
Available in iOS 2.0 and later.

See Also
`+ set` (page 1137)
`+ setWithArray:` (page 1138)
`+ setWithObject:` (page 1138)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 in *objects*, it is added only once to the returned set. Each object receives a `retain` (page 1638) 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 iOS 2.0 and later.

See Also

`+ set` (page 1137)
`+ setWithArray:` (page 1138)
`+ setWithObject:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 1638) message as it is added to the new set.

Return Value

A new set containing the objects from *aSet*.

Availability

Available in iOS 2.0 and later.

See Also

`+ set` (page 1137)
`+ setWithArray:` (page 1138)
`+ setWithObject:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObject:` (page 1154)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 1600).

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 1605) 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 1600).

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 iOS 2.0 and later.

See Also
`- removeObserver:forKeyPath:` (page 1153)

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 iOS 2.0 and later.

See Also
- `anyObject` (page 1142)
- `objectEnumerator` (page 1152)

Related Sample Code
aurioTouch

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 iOS 2.0 and later.

See Also
- `allObjects` (page 1141)
- `objectEnumerator` (page 1152)

Related Sample Code
aurioTouch
GKTank
MoviePlayer
ScrollViewSuite

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 iOS 2.0 and later.

See Also
- member: (page 1151)

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 iOS 2.0 and later.

Related Sample Code
ScrollViewSuite

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 iOS 2.0 and later.

See Also
- descriptionWithLocale: (page 1143)

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 1631) instead.

Availability

Available in iOS 2.0 and later.

See Also

`- description` (page 1143)

Declared In

`NSSet.h`

enumerateObjectsUsingBlock:

Executes a given Block using each object in the receiver.

```
- (void)enumerateObjectsUsingBlock:(void (^)(id obj, BOOL *stop))block
```

Parameters

block

The Block to apply to elements in the set.

The Block takes two arguments:

obj

The element in the set.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Availability

Available in iOS 4.0 and later.

Declared In

`NSSet.h`

enumerateObjectsWithOptions:usingBlock:

Executes a given Block using each object in the receiver, using the specified enumeration options.

```
- (void)enumerateObjectsWithOptions:(NSEnumerationOptions)opts usingBlock:(void
    (^)(id obj, BOOL *stop))block
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

block

The Block to apply to elements in the set.

The Block takes two arguments:

obj

The element in the set.

stop

A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Availability
Available in iOS 4.0 and later.

Declared In
`NSSet.h`

filteredSetUsingPredicate:

Evaluates a given predicate against each object in the receiver and returns a new set containing the objects for which the predicate returns true.

```
- (NSSet *)filteredSetUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate

A predicate.

Return Value
A new set containing the objects in the receiver for which *predicate* returns true.

Discussion
The following example illustrates the use of this method.

```
NSSet *sourceSet =
    [NSSet setWithObjects:@"One", @"Two", @"Three", @"Four", nil];
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith 'T'"];
NSSet *filteredSet =
    [sourceSet filteredSetUsingPredicate:predicate];
// filteredSet contains (Two, Three)
```

Availability
Available in iOS 3.0 and later.

Declared In
NSPredicate.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 1638) message as it is added to the set.

Return Value
An initialized object, which might be different than the original receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithObjects:` (page 1146)
- `initWithObjects:count:` (page 1147)
- `initWithSet:` (page 1147)
- `initWithSet:copyItems:` (page 1148)
+ `setWithArray:` (page 1138)

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.

firstObj, ...

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 1638) message as it is added to the set

Return Value
An initialized object, which might be different than the original receiver.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithArray:` (page 1146)
- `initWithObjects:count:` (page 1147)
- `initWithSet:` (page 1147)
- `initWithSet:copyItems:` (page 1148)
+ `setWithObjects:` (page 1139)

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 in *objects*, it is added only once to the returned set. Each object receives a `retain` (page 1638) 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 iOS 2.0 and later.

See Also
- `initWithArray:` (page 1146)
- `initWithObjects:` (page 1146)
- `initWithSet:` (page 1147)
- `initWithSet:copyItems:` (page 1148)
+ `setWithObjects:count:` (page 1139)

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 iOS 2.0 and later.

See Also

- `initWithArray:` (page 1146)
- `initWithObjects:` (page 1146)
- `initWithObjects:count:` (page 1147)
- `initWithSet:copyItems:` (page 1148)
+ `setWithSet:` (page 1140)

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 1554) 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 iOS 2.0 and later.

See Also

- `initWithArray:` (page 1146)
- `initWithObjects:` (page 1146)
- `initWithObjects:count:` (page 1147)
- `initWithSet:` (page 1147)
+ `setWithSet:` (page 1140)

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 iOS 2.0 and later.

See Also
- isEqualToSet: (page 1149)
- isSubsetOfSet: (page 1150)

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 iOS 2.0 and later.

See Also
- intersectsSet: (page 1149)
- isEqual: (page 1632) (NSObject protocol)
- isSubsetOfSet: (page 1150)

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 iOS 2.0 and later.

See Also
- intersectsSet: (page 1149)
- isEqualToSet: (page 1149)

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 iOS 2.0 and later.

See Also
- makeObjectsPerformSelector:withObject: (page 1151)

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 iOS 2.0 and later.

See Also

`- makeObjectsPerformSelector:` (page 1150)

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 1632)) 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 iOS 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 1141) method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

Availability
Available in iOS 2.0 and later.

See Also
`- nextObject` (page 424) (`NSEnumerator`)

Declared In
`NSSet.h`

objectsPassingTest:

Returns a set of object that pass a test in a given Block.

```
- (NSSet *)objectsPassingTest:(BOOL (^)(id obj, BOOL *stop))predicate
```

Parameters
predicate
The block to apply to elements in the array.

The block takes three arguments:

obj
The element in the set.

stop
A reference to a Boolean value. The block can set the value to `YES` to stop further processing of the set. The `stop` argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
An NSSet containing objects that pass the test.

Availability
Available in iOS 4.0 and later.

Declared In
NSSet.h

objectsWithOptions:passingTest:

Returns a set of object that pass a test in a given Block, using the specified enumeration options.

```
- (NSSet *)objectsWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id
    obj, BOOL *stop))predicate
```

Parameters

opts

A bitmask that specifies the options for the enumeration (whether it should be performed concurrently and whether it should be performed in reverse order).

predicate

The Block to apply to elements in the set.

The Block takes two arguments:

obj

The element in the set.

stop

A reference to a Boolean value. The block can set the value to YES to stop further processing of the set. The stop argument is an out-only argument. You should only ever set this Boolean to YES within the Block.

The Block returns a Boolean value that indicates whether *obj* passed the test.

Return Value
An NSSet containing objects that pass the test.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also

`- addObserver:forKeyPath:options:context:` (page 1141)

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 iOS 2.0 and later.

See Also

`+ set` (page 1137)
`+ setWithArray:` (page 1138)
`+ setWithObject:` (page 1138)
`+ setWithObjects:` (page 1139)
`- setByAddingObjectsFromSet:` (page 1155)
`- setByAddingObjectsFromArray:` (page 1154)

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 iOS 2.0 and later.

See Also
+ set (page 1137)
+ setWithArray: (page 1138)
+ setWithObject: (page 1138)
+ setWithObjects: (page 1139)
- setByAddingObject: (page 1154)
- setByAddingObjectsFromSet: (page 1155)

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 iOS 2.0 and later.

See Also
+ set (page 1137)
+ setWithArray: (page 1138)
+ setWithObject: (page 1138)
+ setWithObjects: (page 1139)
- setByAddingObject: (page 1154)
- setByAddingObjectsFromArray: (page 1154)

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 iOS 2.0 and later.

See Also
- valueForKey: (page 1156)

Declared In
NSKeyValueCoding.h

sortedArrayUsingDescriptors:

Returns an array of the receiver's content sorted as specified by a given array of sort descriptors.

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

Parameters

sortDescriptors

An array of NSSortDescriptor objects.

Return Value
An NSArray containing the receiver's 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 iOS 4.0 and later.

Declared In
NSSortDescriptor.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 iOS 2.0 and later.

See Also
`- setValue:forKey:` (page 1155)

Declared In
`NSKeyValueCoding.h`

NSSortDescriptor Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSSortDescriptor.h
Companion guide	Sort Descriptor Programming Topics

Overview

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 1163))
- 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

Tasks

Initializing a Sort Descriptor

`+ sortDescriptorWithKey:ascending:` (page 1161)
Creates and returns an `NSSortDescriptor` with the specified key and ordering.

`- initWithKey:ascending:` (page 1164)
Returns an `NSSortDescriptor` object initialized with a given property key path and sort order, and with the default comparison selector.

`+ sortDescriptorWithKey:ascending:selector:` (page 1162)
Creates an NSSortDescriptor with the specified ordering and comparison selector.

`- initWithKey:ascending:selector:` (page 1165)
Returns an `NSSortDescriptor` object initialized with a given property key path, sort order, and comparison selector.

`+ sortDescriptorWithKey:ascending:comparator:` (page 1161)
Creates and returns an `NSSortDescriptor` object initialized to do with the given ordering and comparator block.

`- initWithKey:ascending:comparator:` (page 1164)
Returns an `NSSortDescriptor` object initialized to do with the given ordering and comparator block.

Getting Information About a Sort Descriptor

`- ascending` (page 1163)
Returns a Boolean value that indicates whether the receiver specifies sorting in ascending order.

`- key` (page 1166)
Returns the receiver's property key path.

`- selector` (page 1166)
Returns the selector the receiver specifies to use when comparing objects.

Using Sort Descriptors

- `compareObject:toObject:` (page 1163)
  Returns an `NSComparisonResult` value that indicates the ordering of two given objects.
- `reversedSortDescriptor` (page 1166)
  Returns a copy of the receiver with the sort order reversed.

Create an NSComparator for the Sort Descriptor.

- `comparator` (page 1163)
  Creates and returns an `NSComparator` for the sort descriptor.

Class Methods

sortDescriptorWithKey:ascending:

Creates and returns an `NSSortDescriptor` with the specified key and ordering.

```
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending
```

Parameters

key
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` initialized with the specified key and ordering.

Availability
Available in iOS 4.0 and later.

See Also
- `initWithKey:ascending:` (page 1164)

Declared In
`NSSortDescriptor.h`

sortDescriptorWithKey:ascending:comparator:

Creates and returns an `NSSortDescriptor` object initialized to do with the given ordering and comparator block.

```
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending
    comparator:(NSComparator)cmptr
```

Parameters

key

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`.

cmptr

A comparator block.

Return Value

An `NSSortDescriptor` initialized with the specified key, ordering and comparator.

Availability

Available in iOS 4.0 and later.

See Also

`- initWithKey:ascending:comparator:` (page 1164)

Declared In

`NSSortDescriptor.h`

sortDescriptorWithKey:ascending:selector:

Creates an NSSortDescriptor with the specified ordering and comparison selector.

```
+ (id)sortDescriptorWithKey:(NSString *)key ascending:(BOOL)ascending
    selector:(SEL)selector
```

Parameters

key

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 iOS 4.0 and later.

See Also

`- initWithKey:ascending:selector:` (page 1165)

Declared In
NSSortDescriptor.h

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 iOS 2.0 and later.

Declared In
NSSortDescriptor.h

comparator

Creates and returns an NSComparator for the sort descriptor.

```
- (NSComparator)comparator
```

Return Value
An NSComparator object representing the sort descriptor.

Availability
Available in iOS 4.0 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 1166).

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 1166).

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 1166), the values of the properties specified by key (page 1166) of *object1* and *object2*.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- initWithKey:ascending:selector: (page 1165)

Declared In
NSSortDescriptor.h

initWithKey:ascending:comparator:

Returns an NSSortDescriptor object initialized to do with the given ordering and comparator block.

```
- (id)initWithKey:(NSString *)key ascending:(BOOL)ascending
    comparator:(NSComparator)cmptr
```

Parameters

key

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`.

cmptr

A comparator block.

Return Value

An `NSSortDescriptor` initialized with the specified key, ordering and comparator.

Availability

Available in iOS 4.0 and later.

See Also

`+ sortDescriptorWithKey:ascending:comparator:` (page 1161)

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 iOS 2.0 and later.

See Also

`- initWithKey:ascending:` (page 1164)

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa
Related sample code	CryptoExercise WiTap

Overview

`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 1170) and `close` (page 1169)

  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 1170) and `setDelegate:` (page 1172)

  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 ""Delegation" in *Cocoa Fundamentals Guide*" in *Cocoa Fundamentals Guide*.

- `scheduleInRunLoop:forMode:` (page 1171) and `removeFromRunLoop:forMode:` (page 1171)

  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 1643) messages to its delegate.

- `propertyForKey:` (page 1170) and `setProperty:forKey:` (page 1172)

  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 1173) and `streamError` (page 1173)

  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 1170)
  Returns the receiver's property for a given key.
- `setProperty:forKey:` (page 1172)
  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 1170)
  Returns the receiver's delegate.
- `setDelegate:` (page 1172)
  Sets the receiver's delegate.

Using Streams

- `open` (page 1170)
  Opens the receiving stream.
- `close` (page 1169)
  Closes the receiver.

Managing Run Loops

- `scheduleInRunLoop:forMode:` (page 1171)
  Schedules the receiver on a given run loop in a given mode.
- `removeFromRunLoop:forMode:` (page 1171)
  Removes the receiver from a given run loop running in a given mode.

Getting Stream Information

- `streamStatus` (page 1173)
  Returns the receiver's status.
- `streamError` (page 1173)
  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 iOS 2.0 and later.

See Also
- `open` (page 1170)

Declared In
`NSStream.h`

delegate

Returns the receiver's delegate.

```
- (id < NSStreamDelegate >)delegate
```

Return Value
The receiver's delegate. The delegate must implement the `NSStreamDelegate` Protocol.

Discussion
By default, a stream is its own delegate, and subclasses of `NSInputStream` and `NSOutputStream` must maintain this contract.

Availability
Available in iOS 2.0 and later.

See Also
`- setDelegate:` (page 1172)

Related Sample Code
WiTap

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 iOS 2.0 and later.

See Also
`- close` (page 1169)

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 1173) 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 iOS 2.0 and later.

See Also
- `setProperty:forKey:` (page 1172)

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 iOS 2.0 and later.

See Also
- `scheduleInRunLoop:forMode:` (page 1171)

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 iOS 2.0 and later.

See Also
- `removeFromRunLoop:forMode:` (page 1171)

Declared In
`NSStream.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id < NSStreamDelegate >)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 ""Delegation" in *Cocoa Fundamentals Guide*" in *Cocoa Fundamentals Guide*.

Availability
Available in iOS 2.0 and later.

See Also
- `delegate` (page 1170)

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 1173) 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 iOS 2.0 and later.

See Also
- `propertyForKey:` (page 1170)

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 iOS 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 1173) for a description of the available `NSStreamStatus` constants.

Availability
Available in iOS 2.0 and later.

Declared In
`NSStream.h`

Constants

NSStreamStatus

The type declared for the constants listed in "Stream Status Constants" (page 1174).

```
typedef NSUInteger NSStreamStatus;
```

Availability
Available in iOS 2.0 and later.

Declared In
NSStream.h

Stream Status Constants

These constants indicate the current status of a stream. They are returned by `streamStatus` (page 1173).

```
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. This status is returned before the underlying call to open a stream but after it's been created.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusOpening`
The stream is in the process of being opened for reading or for writing. For network streams, this status might include the time after the stream was opened, but while network DNS resolution is happening.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusOpen`
The stream is open, but no reading or writing is occurring.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusReading`
Data is being read from the stream. This status would be returned if code on another thread were to call `streamStatus` (page 1173) on the stream while a `read:maxLength:` (page 646) call (`NSInputStream`) was in progress.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusWriting`
Data is being written to the stream. This status would be returned if code on another thread were to call `streamStatus` (page 1173) on the stream while a `write:maxLength:` (page 1027) call (`NSOutputStream`) was in progress.

Available in iOS 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. When this status is returned, the stream is in a "non-blocking" mode and no data are available.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusClosed`
The stream is closed (`close` (page 1169) has been called on it).

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamStatusError`
The remote end of the connection can't be contacted, or the connection has been severed for some other reason.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

NSStreamEvent

The type declared for the constants listed in "Stream Event Constants" (page 1175).

```
typedef NSUInteger NSStreamEvent;
```

Availability
Available in iOS 2.0 and later.

Declared In
`NSStream.h`

Stream Event Constants

One or more of these constants may be sent to the delegate as a bit field in the second parameter of stream:handleEvent:.

```
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 iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamEventOpenCompleted`
The open has completed successfully.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamEventHasBytesAvailable`
The stream has bytes to be read.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamEventHasSpaceAvailable`
The stream can accept bytes for writing.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamEventErrorOccurred`
An error has occurred on the stream.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamEventEndEncountered`
The end of the stream has been reached.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

NSStream Property Keys

`NSStream` defines these string constants as keys for accessing stream properties using `propertyForKey:` (page 1170) and setting properties with `setProperty:forKey:` (page 1172):

```
NSString * const NSStreamSocketSecurityLevelKey;
NSString * const NSStreamSOCKSProxyConfigurationKey;
NSString * const NSStreamSOCKSProxyHostKey;
NSString * const NSStreamSOCKSProxyPortKey;
NSString * const NSStreamSOCKSProxyVersionKey;
NSString * const NSStreamSOCKSProxyUserKey;
NSString * const NSStreamSOCKSProxyPasswordKey;
NSString * const NSStreamSOCKSProxyVersion4;
NSString * const NSStreamSOCKSProxyVersion5;
NSString * const NSStreamDataWrittenToMemoryStreamKey;
NSString * const NSStreamFileCurrentOffsetKey;
NSString * const NSStreamNetworkServiceType;
```

Constants

`NSStreamSocketSecurityLevelKey`
The security level of the target stream. See "Secure-Socket Layer (SSL) Security Level" (page 1177) for a list of possible values.

Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamFileCurrentOffsetKey`
Value is an `NSNumber` object containing the current absolute offset of the stream.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamNetworkServiceType`
The type of service for the stream. Providing the service type allows the system to properly handle certain attributes of the stream, including routing and suspension behavior. Most streams do not need to set this property. See "Stream Service Types" (page 1179) for a list of possible values.

Available in iOS 4.0 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 1173):

```
NSString * const NSStreamSocketSSLErrorDomain  ;
NSString * const NSStreamSOCKSErrorDomain  ;
```

Constants
`NSStreamSocketSSLErrorDomain`
The error domain used by `NSError` when reporting SSL errors.

Available in iOS 2.0 and later.

Declared in `NSStream.h`.

`NSStreamSOCKSErrorDomain`
The error domain used by `NSError` when reporting SOCKS errors.

Available in iOS 2.0 and later.

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSStream.h`.

Discussion

You access and set these values using the `NSStreamSocketSecurityLevelKey` property key.

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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyUserKey`
Value is an `NSString` object containing the user's name.
Available in iOS 2.0 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyPasswordKey`
Value is an `NSString` object containing the user's password.
Available in iOS 2.0 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyVersion4`
Possible value for `NSStreamSOCKSProxyVersionKey`.
Available in iOS 2.0 and later.
Declared in `NSStream.h`.

`NSStreamSOCKSProxyVersion5`
Possible value for `NSStreamSOCKSProxyVersionKey`.
Available in iOS 2.0 and later.
Declared in `NSStream.h`.

Discussion
You set the dictionary object as the current SOCKS proxy configuration using the `NSStreamSOCKSProxyConfigurationKey` key

Stream Service Types

`NSStream` defines these string constants for specifying the service type of a stream.

```
NSString * const NSStreamNetworkServiceTypeVoIP
```

Constants

`NSStreamNetworkServiceTypeVoIP`

Specifies that the stream is providing VoIP service.

Available in iOS 4.0 and later.

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 iOS 2.0 and later.
Declared in	Foundation/NSString.h Foundation/NSPathUtilities.h Foundation/NSURL.h
Companion guides	String Programming Guide Property List Programming Guide
Related sample code	BonjourWeb CryptoExercise GKRocket ScrollViewSuite WiTap

Overview

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 1246) and `characterAtIndex:` (page 1207)—that provide the basis for all other methods in its interface. The `length` (page 1246) method returns the total number of Unicode characters in the string. `characterAtIndex:` (page 1207) 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 Application Kit Additions Reference, 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.

Understanding characters

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 1246) method and can retrieve a specific character with the `characterAtIndex:` (page 1207) 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/. See also Characters and Grapheme Clusters in *String Programming Guide*.

Interpreting UTF-16-encoded data

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.

Distributed objects

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 1183)).

Make sure your reasons for subclassing `NSString` 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 1184)). 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 1246) and `characterAtIndex:` (page 1207). 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 1222) 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 971) 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 1228) 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating and Initializing Strings

`+ string` (page 1197)
Returns an empty string.

`- init` (page 1230)
Returns an initialized `NSString` object that contains no characters.

`- initWithBytes:length:encoding:` (page 1230)
Returns an initialized `NSString` object containing a given number of bytes from a given buffer of bytes interpreted in a given encoding.

`- initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1231)
Returns an initialized `NSString` object that contains a given number of bytes from a given buffer of bytes interpreted in a given encoding, and optionally frees the buffer.

`- initWithCharacters:length:` (page 1231)
Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.

- `initWithCharactersNoCopy:length:freeWhenDone:` (page 1232)
  Returns an initialized `NSString` object that contains a given number of characters from a given C array of Unicode characters.
- `initWithString:` (page 1242)
  Returns an `NSString` object initialized by copying the characters from another given string.
- `initWithCString:encoding:` (page 1236)
  Returns an `NSString` object initialized using the characters in a given C array, interpreted according to a given encoding.
- `initWithUTF8String:` (page 1242)
  Returns an `NSString` object initialized by copying the characters a given C array of UTF8-encoded bytes.
- `initWithFormat:` (page 1239)
  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 1239)
  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 1240)
  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 1241)
  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 1238)
  Returns an `NSString` object initialized by converting given data into Unicode characters using a given encoding.
+ `stringWithFormat:` (page 1203)
  Returns a string created by using a given format string as a template into which the remaining argument values are substituted.
+ `localizedStringWithFormat:` (page 1196)
  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 1198)
  Returns a string containing a given number of characters taken from a given C array of Unicode characters.
+ `stringWithString:` (page 1204)
  Returns a string created by copying the characters from another given string.
+ `stringWithCString:encoding:` (page 1202)
  Returns a string containing the bytes in a given C array, interpreted according to a given encoding.
+ `stringWithUTF8String:` (page 1204)
  Returns a string created by copying the data from a given C array of UTF8-encoded bytes.
+ `stringWithCString:` (page 1202) Deprecated in iOS 2.0
  Creates a new string using a given C-string. (Deprecated. Use `stringWithCString:encoding:` (page 1202) instead.)

+ `stringWithCString:length:` (page 1203) Deprecated in iOS 2.0
  Returns a string containing the characters in a given C-string. (Deprecated. Use `stringWithCString:encoding:` (page 1202) instead.)

- `initWithCString:` (page 1236) Deprecated in iOS 2.0
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use `initWithCString:encoding:` (page 1236) instead.)

- `initWithCString:length:` (page 1237) Deprecated in iOS 2.0
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use initWithCString:encoding: (page 1236) instead.)

- `initWithCStringNoCopy:length:freeWhenDone:` (page 1238) Deprecated in iOS 2.0
  Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated. Use `initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1231) instead.)

Creating and Initializing a String from a File

+ `stringWithContentsOfFile:encoding:error:` (page 1199)
  Returns a string created by reading data from the file at a given path interpreted using a given encoding.

- `initWithContentsOfFile:encoding:error:` (page 1233)
  Returns an `NSString` object initialized by reading data from the file at a given path using a given encoding.

+ `stringWithContentsOfFile:usedEncoding:error:` (page 1199)
  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 1234)
  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.

+ `stringWithContentsOfFile:` (page 1198) Deprecated in iOS 2.0
  Returns a string created by reading data from the file named by a given path. (Deprecated. Use `stringWithContentsOfFile:encoding:error:` (page 1199) or stringWithContentsOfFile:usedEncoding:error: (page 1199) instead.)

- `initWithContentsOfFile:` (page 1233) Deprecated in iOS 2.0
  Initializes the receiver, a newly allocated `NSString` object, by reading data from the file named by *`path`*. (Deprecated. Use `initWithContentsOfFile:encoding:error:` (page 1233) or `initWithContentsOfFile:usedEncoding:error:` (page 1234) instead.)

Creating and Initializing a String from an URL

+ `stringWithContentsOfURL:encoding:error:` (page 1200)
  Returns a string created by reading data from a given URL interpreted using a given encoding.

- `initWithContentsOfURL:encoding:error:` (page 1235)
  Returns an `NSString` object initialized by reading data from a given URL interpreted using a given encoding.

+ stringWithContentsOfURL:usedEncoding:error: (page 1201)
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 1235)
Returns an NSString object initialized by reading data from a given URL and returns by reference the encoding used to interpret the data.

+ stringWithContentsOfURL: (page 1200) Deprecated in iOS 2.0
Returns a string created by reading data from the file named by a given URL. (Deprecated. Use stringWithContentsOfURL:encoding:error: (page 1200) or stringWithContentsOfURL:usedEncoding:error: (page 1201) instead.)

- initWithContentsOfURL: (page 1234) Deprecated in iOS 2.0
Initializes the receiver, a newly allocated NSString object, by reading data from the location named by a given URL. (Deprecated. Use initWithContentsOfURL:encoding:error: (page 1235) or initWithContentsOfURL:usedEncoding:error: (page 1235) instead.)

Writing to a File or URL

- writeToFile:atomically:encoding:error: (page 1277)
Writes the contents of the receiver to a file at a given path using a given encoding.

- writeToURL:atomically:encoding:error: (page 1278)
Writes the contents of the receiver to the URL specified by *url* using the specified encoding.

- writeToFile:atomically: (page 1276) Deprecated in iOS 2.0
Writes the contents of the receiver to the file specified by a given path. (Deprecated. Use writeToFile:atomically:encoding:error: (page 1277) instead.)

- writeToURL:atomically: (page 1278) Deprecated in iOS 2.0
Writes the contents of the receiver to the location specified by a given URL. (Deprecated. Use writeToURL:atomically:encoding:error: (page 1278) instead.)

Getting a String's Length

- length (page 1246)
Returns the number of Unicode characters in the receiver.

- lengthOfBytesUsingEncoding: (page 1246)
Returns the number of bytes required to store the receiver in a given encoding.

- maximumLengthOfBytesUsingEncoding: (page 1250)
Returns the maximum number of bytes needed to store the receiver in a given encoding.

Getting Characters and Bytes

- characterAtIndex: (page 1207)
Returns the character at a given array position.

- getCharacters:range: (page 1222)
Copies characters from a given range in the receiver into a given buffer.

- `getBytes:maxLength:usedLength:encoding:options:range:remainingRange:` (page 1221)
  Gets a given range of characters as bytes in a specified encoding.
- `getCharacters:` (page 1222) Deprecated in iOS 4.0
  Copies all characters from the receiver into a given buffer. (Deprecated. This method is unsafe because it could potentially cause buffer overruns. Use `getCharacters:range:` (page 1222) instead.)

Getting C Strings

- `cStringUsingEncoding:` (page 1215)
  Returns a representation of the receiver as a C string using a given encoding.
- `getCString:maxLength:encoding:` (page 1224)
  Converts the receiver's content to a given encoding and stores them in a buffer.
- `UTF8String` (page 1276)
  Returns a null-terminated UTF8 representation of the receiver.
- `cString` (page 1214) Deprecated in iOS 2.0
  Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated. Use `cStringUsingEncoding:` (page 1215) or `UTF8String` (page 1276) instead.)
- `cStringLength` (page 1214) Deprecated in iOS 2.0
  Returns the length in `char`-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated. Use lengthOfBytesUsingEncoding: (page 1246) or maximumLengthOfBytesUsingEncoding: (page 1250) instead.)
- `getCString:` (page 1223) Deprecated in iOS 2.0
  Invokes `getCString:maxLength:range:remainingRange:` (page 1225) with `NSMaximumStringLength` as the maximum length, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use cStringUsingEncoding: (page 1215) or `dataUsingEncoding:allowLossyConversion:` (page 1216) instead.)
- `getCString:maxLength:` (page 1224) Deprecated in iOS 2.0
  Invokes `getCString:maxLength:range:remainingRange:` (page 1225) with *`maxLength`* as the maximum length in `char`-sized units, the receiver's entire extent as the range, and `NULL` for the remaining range. (Deprecated. Use getCString:maxLength:encoding: (page 1224) instead.)
- `getCString:maxLength:range:remainingRange:` (page 1225) Deprecated in iOS 2.0
  Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated. Use getCString:maxLength:encoding: (page 1224) instead.)
- `lossyCString` (page 1249) Deprecated in iOS 2.0
  Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated. Use cStringUsingEncoding: (page 1215) or `dataUsingEncoding:allowLossyConversion:` (page 1216) instead.)

Combining Strings

- `stringByAppendingFormat:` (page 1263)
  Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
- `stringByAppendingString:` (page 1265)
  Returns a new string made by appending a given string to the receiver.

- `stringByPaddingToLength:withString:startingAtIndex:` (page 1268)
  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 1213)
  Returns an array containing substrings from the receiver that have been divided by a given separator.
- `componentsSeparatedByCharactersInSet:` (page 1212)
  Returns an array containing substrings from the receiver that have been divided by characters in a given set.
- `stringByTrimmingCharactersInSet:` (page 1273)
  Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
- `substringFromIndex:` (page 1274)
  Returns a new string containing the characters of the receiver from the one at a given index to the end.
- `substringWithRange:` (page 1275)
  Returns a string object containing the characters of the receiver that lie within a given range.
- `substringToIndex:` (page 1274)
  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 1255)
  Finds and returns the range in the receiver of the first character from a given character set.
- `rangeOfCharacterFromSet:options:` (page 1255)
  Finds and returns the range in the receiver of the first character, using given options, from a given character set.
- `rangeOfCharacterFromSet:options:range:` (page 1256)
  Finds and returns the range in the receiver of the first character from a given character set found in a given range with given options.
- `rangeOfString:` (page 1258)
  Finds and returns the range of the first occurrence of a given string within the receiver.
- `rangeOfString:options:` (page 1259)
  Finds and returns the range of the first occurrence of a given string within the receiver, subject to given options.
- `rangeOfString:options:range:` (page 1259)
  Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.
- `rangeOfString:options:range:locale:` (page 1260)
  Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

- `enumerateLinesUsingBlock:` (page 1218)
  Enumerates all the lines in a string.
- `enumerateSubstringsInRange:options:usingBlock:` (page 1219)
  Enumerates the substrings of the specified type in the specified range of the string.

Replacing Substrings

- `stringByReplacingOccurrencesOfString:withString:` (page 1270)
  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 1270)
  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 1269)
  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 1227)
  Returns by reference the beginning of the first line and the end of the last line touched by the given range.
- `lineRangeForRange:` (page 1247)
  Returns the range of characters representing the line or lines containing a given range.
- `getParagraphStart:end:contentsEnd:forRange:` (page 1228)
  Returns by reference the beginning of the first paragraph and the end of the last paragraph touched by the given range.
- `paragraphRangeForRange:` (page 1251)
  Returns the range of characters representing the paragraph or paragraphs containing a given range.

Determining Composed Character Sequences

- `rangeOfComposedCharacterSequenceAtIndex:` (page 1257)
  Returns the range in the receiver of the composed character sequence located at a given index.
- `rangeOfComposedCharacterSequencesForRange:` (page 1258)
  Returns the range in the receiver of the composed character sequences in a given range.

Converting String Contents Into a Property List

- `propertyList` (page 1253)
  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 1254)
  Returns a dictionary object initialized with the keys and values found in the receiver.

Identifying and Comparing Strings

- `caseInsensitiveCompare:` (page 1207)
  Returns the result of invoking `compare:options:` (page 1209) with `NSCaseInsensitiveSearch` as the only option.
- `localizedCaseInsensitiveCompare:` (page 1247)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and a given string using a case-insensitive, localized, comparison.
- `compare:` (page 1208)
  Returns the result of invoking `compare:options:range:` (page 1210) with no options and the receiver's full extent as the range.
- `localizedCompare:` (page 1248)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of the receiver and another given string using a localized comparison.
- `compare:options:` (page 1209)
  Returns the result of invoking `compare:options:range:` (page 1210) with a given mask as the options and the receiver's full extent as the range.
- `compare:options:range:` (page 1210)
  Returns the result of invoking `compare:options:range:locale:` (page 1211) with a `nil` locale.
- `compare:options:range:locale:` (page 1211)
  Returns an `NSComparisonResult` value that indicates the lexical ordering of a specified range within the receiver and a given string.
- `localizedStandardCompare:` (page 1248)
  Compares strings as sorted by the Finder.
- `hasPrefix:` (page 1229)
  Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
- `hasSuffix:` (page 1229)
  Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
- `isEqualToString:` (page 1244)
  Returns a Boolean value that indicates whether a given string is equal to the receiver using an literal Unicode-based comparison.
- `hash` (page 1228)
  Returns an unsigned integer that can be used as a hash table address.

Folding Strings

- `stringByFoldingWithOptions:locale:` (page 1268)
  Returns a string with the given character folding options applied.

Getting a Shared Prefix

- `commonPrefixWithString:options:` (page 1208)
  Returns a string containing characters the receiver and a given string have in common, starting from the beginning of each up to the first characters that aren't equivalent.

Changing Case

- `capitalizedString` (page 1206)
  Returns a capitalized representation of the receiver.
- `lowercaseString` (page 1249)
  Returns lowercased representation of the receiver.
- `uppercaseString` (page 1276)
  Returns an uppercased representation of the receiver.

Getting Strings with Mapping

- `decomposedStringWithCanonicalMapping` (page 1217)
  Returns a string made by normalizing the receiver's contents using Form D.
- `decomposedStringWithCompatibilityMapping` (page 1217)
  Returns a string made by normalizing the receiver's contents using Form KD.
- `precomposedStringWithCanonicalMapping` (page 1253)
  Returns a string made by normalizing the receiver's contents using Form C.
- `precomposedStringWithCompatibilityMapping` (page 1253)
  Returns a string made by normalizing the receiver's contents using Form KC.

Getting Numeric Values

- `doubleValue` (page 1218)
  Returns the floating-point value of the receiver's text as a `double`.
- `floatValue` (page 1221)
  Returns the floating-point value of the receiver's text as a `float`.
- `intValue` (page 1243)
  Returns the integer value of the receiver's text.
- `integerValue` (page 1243)
  Returns the `NSInteger` value of the receiver's text.
- `longLongValue` (page 1248)
  Returns the `long long` value of the receiver's text.
- `boolValue` (page 1205)
  Returns the Boolean value of the receiver's text.

Working with Encodings

`+ availableStringEncodings` (page 1194)
Returns a zero-terminated list of the encodings string objects support in the application's environment.

`+ defaultCStringEncoding` (page 1195)
Returns the C-string encoding assumed for any method accepting a C string as an argument.

`+ localizedNameOfStringEncoding:` (page 1195)
Returns a human-readable string giving the name of a given encoding.

`- canBeConvertedToEncoding:` (page 1206)
Returns a Boolean value that indicates whether the receiver can be converted to a given encoding without loss of information.

`- dataUsingEncoding:` (page 1216)
Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

`- dataUsingEncoding:allowLossyConversion:` (page 1216)
Returns an `NSData` object containing a representation of the receiver encoded using a given encoding.

`- description` (page 1217)
Returns the receiver.

`- fastestEncoding` (page 1220)
Returns the fastest encoding to which the receiver may be converted without loss of information.

`- smallestEncoding` (page 1261)
Returns the smallest encoding to which the receiver can be converted without loss of information.

Working with Paths

`+ pathWithComponents:` (page 1197)
Returns a string built from the strings in a given array by concatenating them with a path separator between each pair.

`- pathComponents` (page 1251)
Returns an array of `NSString` objects containing, in order, each path component of the receiver.

`- completePathIntoString:caseSensitive:matchesIntoArray:filterTypes:` (page 1212)
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 1220)
Returns a file system-specific representation of the receiver.

`- getFileSystemRepresentation:maxLength:` (page 1226)
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 1244)
Returning a Boolean value that indicates whether the receiver represents an absolute path.

`- lastPathComponent` (page 1245)
Returns the last path component of the receiver.

`- pathExtension` (page 1252)
Interprets the receiver as a path and returns the receiver's extension, if any.

- `stringByAbbreviatingWithTildeInPath` (page 1262)
  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 1263)
  Returns a new string made by appending to the receiver a given string.
- `stringByAppendingPathExtension:` (page 1264)
  Returns a new string made by appending to the receiver an extension separator followed by a given extension.
- `stringByDeletingLastPathComponent` (page 1266)
  Returns a new string made by deleting the last path component from the receiver, along with any final path separator.
- `stringByDeletingPathExtension` (page 1266)
  Returns a new string made by deleting the extension (if any, and only the last) from the receiver.
- `stringByExpandingTildeInPath` (page 1267)
  Returns a new string made by expanding the initial component of the receiver to its full path value.
- `stringByResolvingSymlinksInPath` (page 1271)
  Returns a new string made from the receiver by resolving all symbolic links and standardizing path.
- `stringByStandardizingPath` (page 1272)
  Returns a new string made by removing extraneous path components from the receiver.
- `stringsByAppendingPaths:` (page 1273)
  Returns an array of strings made by separately appending to the receiver each string in in a given array.

Working with URLs

- `stringByAddingPercentEscapesUsingEncoding:` (page 1262)
  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 1271)
  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 1279) 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 iOS 2.0 and later.

See Also
`+ localizedNameOfStringEncoding:` (page 1195)

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:` (page 1202)—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 1283) for a full list of supported encodings.

Availability
Available in iOS 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 iOS 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 1240) 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 iOS 2.0 and later.

See Also
+ stringWithFormat: (page 1203)
- initWithFormat:locale: (page 1240)

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 1272) to resolve empty components, references to the parent directory, and so on.

Availability
Available in iOS 2.0 and later.

See Also
- pathComponents (page 1251)

Declared In
NSPathUtilities.h

string

Returns an empty string.

```
+ (id)string
```

Return Value
An empty string.

Availability
Available in iOS 2.0 and later.

See Also
- init (page 1230)

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 iOS 2.0 and later.

See Also
- initWithCharacters:length: (page 1231)

Declared In
NSString.h

stringWithContentsOfFile:

Returns a string created by reading data from the file named by a given path. (Deprecated in iOS 2.0. Use stringWithContentsOfFile:encoding:error: (page 1199) or stringWithContentsOfFile:usedEncoding:error: (page 1199) instead.)

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

Discussion
If the contents begin with a Unicode byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (EFBBBF), interprets the contents as UTF-8. Otherwise, interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns nil if the file can't be opened.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
+ stringWithContentsOfFile:encoding:error: (page 1199)
+ stringWithContentsOfFile:usedEncoding:error: (page 1199)

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 iOS 2.0 and later.

See Also
- initWithContentsOfFile:encoding:error: (page 1233)

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 iOS 2.0 and later.

See Also
- initWithContentsOfFile:encoding:error: (page 1233)

Declared In
NSString.h

stringWithContentsOfURL:

Returns a string created by reading data from the file named by a given URL. (Deprecated in iOS 2.0. Use stringWithContentsOfURL:encoding:error: (page 1200) or stringWithContentsOfURL:usedEncoding:error: (page 1201) instead.)

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

Discussion
If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters. If the contents begin with a UTF-8 byte-order mark (EFBBBF), interprets the contents as UTF-8. Otherwise interprets the contents as data in the default C string encoding. Since the default C string encoding will vary with the user's configuration, do not depend on this method unless you are using Unicode or UTF-8 or you can verify the default C string encoding. Returns nil if the location can't be opened.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
+ stringWithContentsOfURL:encoding:error: (page 1200)
+ stringWithContentsOfURL:usedEncoding:error: (page 1201)

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 iOS 2.0 and later.

See Also
`+ stringWithContentsOfURL:usedEncoding:error:` (page 1201)
`- initWithContentsOfURL:encoding:error:` (page 1235)

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 iOS 2.0 and later.

See Also
`+ stringWithContentsOfURL:encoding:error:` (page 1200)
`- initWithContentsOfURL:usedEncoding:error:` (page 1235)

Declared In
`NSString.h`

stringWithCString:

Creates a new string using a given C-string. (Deprecated in iOS 2.0. Use `stringWithCString:encoding:` (page 1202) instead.)

```
+ (id)stringWithCString:(const char *)cString
```

Discussion
cString should contain data in the default C string encoding. If the argument passed to `stringWithCString:` is not a zero-terminated C-string, the results are undefined.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
`+ stringWithCString:encoding:` (page 1202)

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 iOS 2.0 and later.

See Also
`- initWithCString:encoding:` (page 1236)

Declared In
`NSString.h`

stringWithCString:length:

Returns a string containing the characters in a given C-string. (Deprecated in iOS 2.0. Use `stringWithCString:encoding:` (page 1202) instead.)

```
+ (id)stringWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion

cString must not be NULL. *cString* should contain characters in the default C-string encoding. This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a `NULL` character.

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 2.0.

See Also

`+ stringWithCString:encoding:` (page 1202)

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 1196), 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 iOS 2.0 and later.

See Also

`- initWithFormat:` (page 1239)

`+ localizedStringWithFormat:` (page 1196)

Related Sample Code
aurioTouch
GKRocket
GKTank
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

See Also
`- initWithString:` (page 1242)

Related Sample Code
BonjourWeb

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 iOS 2.0 and later.

See Also
- initWithString: (page 1242)

Related Sample Code
BonjourWeb
CryptoExercise

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 iOS 2.0 and later.

See Also
- integerValue (page 1243)
- scanInt: (page 1124) (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 iOS 2.0 and later.

See Also

- `dataUsingEncoding:allowLossyConversion:` (page 1216)

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 1227)). 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 1249) for an example.

Availability

Available in iOS 2.0 and later.

See Also

- `lowercaseString` (page 1249)
- `uppercaseString` (page 1276)

Declared In
NSString.h

caseInsensitiveCompare:

Returns the result of invoking compare:options: (page 1209) 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 1209) with NSCaseInsensitiveSearch as the only option.

Discussion
If you are comparing strings to present to the end-user, you should typically use localizedCaseInsensitiveCompare: (page 1247) instead.

Availability
Available in iOS 2.0 and later.

See Also
- localizedCaseInsensitiveCompare: (page 1247)
- compare:options: (page 1209)

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 iOS 2.0 and later.

See Also
- `getCharacters:range:` (page 1222)

Declared In
`NSString.h`

commonPrefixWithString:options:

Returns a string containing characters the receiver and a given string have in common, starting from the beginning of each up to the first characters that aren't equivalent.

```
- (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 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 iOS 2.0 and later.

See Also
- `hasPrefix:` (page 1229)

Declared In
`NSString.h`

compare:

Returns the result of invoking `compare:options:range:` (page 1210) 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 1210) 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 1248) or `localizedCaseInsensitiveCompare:` (page 1247) instead.

Availability
Available in iOS 2.0 and later.

See Also
- `localizedCompare:` (page 1248)
- `localizedCaseInsensitiveCompare:` (page 1247)
- `compare:options:` (page 1209)
- `caseInsensitiveCompare:` (page 1207)
- `isEqualToString:` (page 1244)

Related Sample Code
GKRocket

Declared In
`NSString.h`

compare:options:

Returns the result of invoking `compare:options:range:` (page 1210) 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 details on these options.

Return Value
The result of invoking `compare:options:range:` (page 1210) 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 1248) or `localizedCaseInsensitiveCompare:` (page 1247) instead, or use `compare:options:range:locale:` (page 1211) and pass the user's locale.

Availability
Available in iOS 2.0 and later.

See Also
- `localizedCompare:` (page 1248)
- `localizedCaseInsensitiveCompare:` (page 1247)
- `compare:options:range:locale:` (page 1211)
- `caseInsensitiveCompare:` (page 1207)
- `isEqualToString:` (page 1244)

Declared In
`NSString.h`

compare:options:range:

Returns the result of invoking `compare:options:range:locale:` (page 1211) 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 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 1211) 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 1211) instead and pass the user's locale (`currentLocale` (page 692) [`NSLocale`]).

Availability
Available in iOS 2.0 and later.

See Also
- `localizedCompare:` (page 1248)
- `localizedCaseInsensitiveCompare:` (page 1247)
- `compare:options:` (page 1209)
- `caseInsensitiveCompare:` (page 1207)
- `isEqualToString:` (page 1244)

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 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 692) [NSLocale]).

The locale argument affects both equality and ordering algorithms. For example, in some locales, accented characters are ordered immediately after the base; other locales order them after "z".

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 iOS 2.0 and later.

See Also
- localizedCompare: (page 1248)
- localizedCaseInsensitiveCompare: (page 1247)
- caseInsensitiveCompare: (page 1207)
- compare: (page 1208)

- compare:options: (page 1209)
- compare:options:range: (page 1210)
- isEqualToString: (page 1244)

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 iOS 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 iOS 2.0 and later.

See Also

- `componentsSeparatedByString:` (page 1213)
- `stringByTrimmingCharactersInSet:` (page 1273)

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 iOS 2.0 and later.

See Also
componentsJoinedByString: (page 51) (NSArray)
- pathComponents (page 1251)

Declared In
NSString.h

cString

Returns a representation of the receiver as a C string in the default C-string encoding. (Deprecated in iOS 2.0. Use cStringUsingEncoding: (page 1215) or UTF8String (page 1276) instead.)

```
- (const char *)cString
```

Discussion
The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use getCString: (page 1223) if it needs to store the C string outside of the autorelease context in which the C string is created.

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 1206) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1249) or dataUsingEncoding:allowLossyConversion: (page 1216) to get a C-string representation with some loss of information.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- cStringUsingEncoding: (page 1215)
- getCString:maxLength:encoding: (page 1224)
- UTF8String (page 1276)

Declared In
NSString.h

cStringLength

Returns the length in char-sized units of the receiver's C-string representation in the default C-string encoding. (Deprecated in iOS 2.0. Use lengthOfBytesUsingEncoding: (page 1246) or maximumLengthOfBytesUsingEncoding: (page 1250) instead.)

```
- (NSUInteger)cStringLength
```

Discussion
Raises if the receiver can't be represented in the default C-string encoding without loss of information. You can also use canBeConvertedToEncoding: (page 1206) to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1249) to get a C-string representation with some loss of information, then check its length explicitly using the ANSI function strlen().

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- `lengthOfBytesUsingEncoding:` (page 1246)
- `maximumLengthOfBytesUsingEncoding:` (page 1250)
- `UTF8String` (page 1276)

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 1224) if it needs to store the C string beyond this time.

You can use `canBeConvertedToEncoding:` (page 1206) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use `dataUsingEncoding:allowLossyConversion:` (page 1216) 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 iOS 2.0 and later.

See Also
- `getCString:` (page 1223)
- `canBeConvertedToEncoding:` (page 1206)
+ `defaultCStringEncoding` (page 1195)
- `cStringLength` (page 1214)
- `UTF8String` (page 1276)

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 1216) with `NO` as the second argument (that is, requiring lossless conversion).

Availability

Available in iOS 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 iOS 2.0 and later.

See Also

`+ availableStringEncodings` (page 1194)

`- canBeConvertedToEncoding:` (page 1206)

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 iOS 2.0 and later.

See Also
- precomposedStringWithCanonicalMapping (page 1253)
- decomposedStringWithCompatibilityMapping (page 1217)

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 iOS 2.0 and later.

See Also
- precomposedStringWithCompatibilityMapping (page 1253)
- decomposedStringWithCanonicalMapping (page 1217)

Declared In
NSString.h

description

Returns the receiver.

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

Return Value
The receiver.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `floatValue` (page 1221)
- `longLongValue` (page 1248)
- `integerValue` (page 1243)
- `scanDouble:` (page 1122) (`NSScanner`)

Declared In
NSString.h

enumerateLinesUsingBlock:

Enumerates all the lines in a string.

```
- (void)enumerateLinesUsingBlock:(void (^)(NSString *line, BOOL *stop))block
```

Parameters
block

The block executed for the enumeration.

The block takes two arguments:

line

The to enumerate containing just the contents of the line, without the line terminators.

stop

A reference to a Boolean value that the block can use to stop the enumeration by setting `*stop = YES`; it should not touch `*stop` otherwise.

Availability
Available in iOS 4.0 and later.

Declared In
`NSString.h`

enumerateSubstringsInRange:options:usingBlock:

Enumerates the substrings of the specified type in the specified range of the string.

```
- (void)enumerateSubstringsInRange:(NSRange)range
    options:(NSStringEnumerationOptions)opts usingBlock:(void (^)(NSString
    *substring, NSRange substringRange, NSRange enclosingRange, BOOL *stop))block
```

Parameters

range

The range within the string to enumerate substrings.

opts

Options specifying types of substrings and enumeration styles.

block

The block executed for the enumeration.

The block takes four arguments:

substring

The enumerated string.

substringRange

The range of the enumerated string in the receiver.

enclosingRange

The range that includes the substring as well as any separator or filler characters that follow. For instance, for lines, *enclosingRange* contains the line terminators. The *enclosingRange* for the first string enumerated also contains any characters that occur before the string. Consecutive enclosing ranges are guaranteed not to overlap, and every single character in the enumerated range is included in one and only one enclosing range.

stop

A reference to a Boolean value that the block can use to stop the enumeration by setting `*stop = YES`; it should not touch `*stop` otherwise.

Discussion

If this method is sent to an instance of `NSMutableString`, mutation (deletion, addition, or change) is allowed, as long as it is within *enclosingRange*. After a mutation, the enumeration continues with the range immediately following the processed range, after the length of the processed range is adjusted for the mutation. (The enumerator assumes any change in length occurs in the specified range.)

For example, if the block is called with a range starting at location N, and the block deletes all the characters in the supplied range, the next call will also pass N as the index of the range. This is the case even if mutation of the previous range changes the string in such a way that the following substring would have extended to include the already enumerated range. For example, if the string "Hello World" is enumerated via words, and the block changes "Hello " to "Hello", thus forming "HelloWorld", the next enumeration will return "World" rather than "HelloWorld".

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
- `smallestEncoding` (page 1261)
- `getCharacters:range:` (page 1222)

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 1226).

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 1226) 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 532) method.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `doubleValue` (page 1218)
- `longLongValue` (page 1248)
- `integerValue` (page 1243)
- `scanFloat:` (page 1122) (`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 iOS 2.0 and later.

Declared In
NSString.h

getCharacters:

Copies all characters from the receiver into a given buffer. (Deprecated in iOS 4.0. This method is unsafe because it could potentially cause buffer overruns. Use getCharacters:range: (page 1222) instead.)

```
- (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 1222) with *buffer* and the entire extent of the receiver as the range.

Availability
Available in iOS 2.0 and later.
Deprecated in iOS 4.0.

See Also
- length (page 1246)

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 1207) repeatedly, correctly extracting the characters, though very inefficiently. Subclasses should override it to provide a fast implementation.

Availability
Available in iOS 2.0 and later.

Declared In
NSString.h

getCString:

Invokes getCString:maxLength:range:remainingRange: (page 1225) with NSMaximumStringLength as the maximum length, the receiver's entire extent as the range, and NULL for the remaining range. (Deprecated in iOS 2.0. Use cStringUsingEncoding: (page 1215) or dataUsingEncoding:allowLossyConversion: (page 1216) instead.)

```
- (void)getCString:(char *)buffer
```

Discussion
buffer must be large enough to contain the resulting C-string plus a terminating NULL character (which this method adds—[string cStringLength]).

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 1206) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1249) or dataUsingEncoding:allowLossyConversion: (page 1216) to get a C-string representation with some loss of information.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- cStringUsingEncoding: (page 1215)
- getCString:maxLength:encoding: (page 1224)
- UTF8String (page 1276)

Declared In
NSString.h

getCString:maxLength:

Invokes getCString:maxLength:range:remainingRange: (page 1225) with *maxLength* as the maximum length in char-sized units, the receiver's entire extent as the range, and NULL for the remaining range. (Deprecated in iOS 2.0. Use getCString:maxLength:encoding: (page 1224) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
```

Discussion
buffer must be large enough to contain *maxLength* chars plus a terminating zero char (which this method adds).

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 1206) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1249) or dataUsingEncoding:allowLossyConversion: (page 1216) to get a C-string representation with some loss of information.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- cStringUsingEncoding: (page 1215)
- getCString:maxLength:encoding: (page 1224)
- UTF8String (page 1276)

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: (page 1224) method which it replaces. (The buffer should include room for *maxBufferCount* bytes; this number should accommodate the expected size of the return value plus the NULL termination byte, which this method adds.)

You can use canBeConvertedToEncoding: (page 1206) to check whether a string can be losslessly converted to *encoding*. If it can't, you can use dataUsingEncoding:allowLossyConversion: (page 1216) 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 iOS 2.0 and later.

See Also
- cStringUsingEncoding: (page 1215)
- canBeConvertedToEncoding: (page 1206)
- UTF8String (page 1276)

Declared In
NSString.h

getCString:maxLength:range:remainingRange:

Converts the receiver's content to the default C-string encoding and stores them in a given buffer. (Deprecated in iOS 2.0. Use getCString:maxLength:encoding: (page 1224) instead.)

```
- (void)getCString:(char *)buffer maxLength:(NSUInteger)maxLength
    range:(NSRange)aRange remainingRange:(NSRangePointer)leftoverRange
```

Discussion
buffer must be large enough to contain *maxLength* bytes plus a terminating zero character (which this method adds). Copies and converts as many characters as possible from *aRange* and stores the range of those not converted in the range given by *leftoverRange* (if it's non-nil). Raises an NSRangeException if any part of *aRange* lies beyond the end of the string.

Raises an NSCharacterConversionException if the receiver can't be represented in the default C-string encoding without loss of information. Use canBeConvertedToEncoding: (page 1206) if necessary to check whether a string can be losslessly converted to the default C-string encoding. If it can't, use lossyCString (page 1249) or dataUsingEncoding:allowLossyConversion: (page 1216) to get a C-string representation with some loss of information.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- cStringUsingEncoding: (page 1215)
- getCString:maxLength:encoding: (page 1224)
- UTF8String (page 1276)

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 iOS 2.0 and later.

See Also
- fileSystemRepresentation (page 1220)

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.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability

Available in iOS 2.0 and later.

See Also

- `lineRangeForRange:` (page 1247)
- `substringWithRange:` (page 1275)

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 1227), you can use the results of this method to construct the ranges for paragraphs.

Availability

Available in iOS 2.0 and later.

See Also

`- paragraphRangeForRange:` (page 1251)

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 1244) 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 iOS 2.0 and later.

Related Sample Code
GKTank

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 more information.

Availability
Available in iOS 2.0 and later.

See Also
- `hasSuffix:` (page 1229)
- `compare:options:range:` (page 1210)

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 more information.

Availability
Available in iOS 2.0 and later.

See Also
- `hasPrefix:` (page 1229)
- `compare:options:range:` (page 1210)

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 iOS 2.0 and later.

See Also
`+ string` (page 1197)

Declared In
`NSString.h`

initWithBytes:length:encoding:

Returns an initialized `NSString` object containing a given number of bytes from a given buffer of bytes interpreted in a given encoding.

```
- (id)initWithBytes:(const void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding
```

Parameters

bytes
A buffer of bytes interpreted in the encoding specified by *encoding*.

length
The number of bytes to use from *bytes*.

encoding
The character encoding applied to *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 iOS 2.0 and later.

See Also
- `initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1231)

Declared In
`NSString.h`

initWithBytesNoCopy:length:encoding:freeWhenDone:

Returns an initialized `NSString` object that contains a given number of bytes from a given buffer of bytes interpreted in a given encoding, and optionally frees the buffer.

```
- (id)initWithBytesNoCopy:(void *)bytes length:(NSUInteger)length
    encoding:(NSStringEncoding)encoding freeWhenDone:(BOOL)flag
```

Parameters

bytes
A buffer of bytes interpreted in the encoding specified by *encoding*.

length
The number of bytes to use from *bytes*.

encoding
The character encoding of *bytes*.

flag
If `YES`, the receiver frees 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 iOS 2.0 and later.

See Also
- `initWithBytes:length:encoding:` (page 1230)

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 iOS 2.0 and later.

See Also

+ stringWithCharacters:length: (page 1198)

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 iOS 2.0 and later.

See Also

+ stringWithCharacters:length: (page 1198)

Declared In
NSString.h

initWithContentsOfFile:

Initializes the receiver, a newly allocated NSString object, by reading data from the file named by *path*. (Deprecated in iOS 2.0. Use initWithContentsOfFile:encoding:error: (page 1233) or initWithContentsOfFile:usedEncoding:error: (page 1234) instead.)

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

Discussion
Initializes the receiver, a newly allocated NSString object, by reading data from the file named by *path*. If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or nil if the file can't be opened.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- initWithContentsOfFile:encoding:error: (page 1233)
- initWithContentsOfFile:usedEncoding:error: (page 1234)

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 iOS 2.0 and later.

See Also
+ stringWithContentsOfFile:encoding:error: (page 1199)
- initWithContentsOfFile:usedEncoding:error: (page 1234)

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 iOS 2.0 and later.

See Also
+ stringWithContentsOfFile:encoding:error: (page 1199)
- initWithContentsOfFile:encoding:error: (page 1233)

Declared In
NSString.h

initWithContentsOfURL:

Initializes the receiver, a newly allocated NSString object, by reading data from the location named by a given URL. (Deprecated in iOS 2.0. Use initWithContentsOfURL:encoding:error: (page 1235) or initWithContentsOfURL:usedEncoding:error: (page 1235) instead.)

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

Discussion
Initializes the receiver, a newly allocated NSString object, by reading data from the location named by *aURL*. If the contents begin with a byte-order mark (U+FEFF or U+FFFE), interprets the contents as Unicode characters; otherwise interprets the contents as data in the default C string encoding. Returns an initialized object, which might be different from the original receiver, or nil if the location can't be opened.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- `initWithContentsOfURL:encoding:error:` (page 1235)
- `initWithContentsOfURL:usedEncoding:error:` (page 1235)

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 iOS 2.0 and later.

See Also
- `+ stringWithContentsOfURL:encoding:error:` (page 1200)

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 iOS 2.0 and later.

See Also
`+ stringWithContentsOfURL:usedEncoding:error:` (page 1201)

Declared In
`NSString.h`

initWithCString:

Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in iOS 2.0. Use `initWithCString:encoding:` (page 1236) instead.)

```
- (id)initWithCString:(const char *)cString
```

Discussion
cString must be a zero-terminated C string in the default C string encoding, and may not be `NULL`. Returns an initialized object, which might be different from the original receiver.

To create an immutable string from an immutable C string buffer, do not attempt to use this method. Instead, use `initWithCStringNoCopy:length:freeWhenDone:` (page 1238).

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
`- initWithCString:encoding:` (page 1236)

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 iOS 2.0 and later.

See Also

+ stringWithCString: (page 1202)

- initWithCStringNoCopy:length:freeWhenDone: (page 1238)

+ defaultCStringEncoding (page 1195)

Declared In

NSString.h

initWithCString:length:

Initializes the receiver, a newly allocated NSString object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in iOS 2.0. Use initWithCString:encoding: (page 1236) instead.)

```
- (id)initWithCString:(const char *)cString length:(NSUInteger)length
```

Discussion

This method converts *length* * sizeof(char) bytes from *cString* and doesn't stop short at a zero character. *cString* must contain bytes in the default C-string encoding and may not be NULL. Returns an initialized object, which might be different from the original receiver.

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 2.0.

See Also

- initWithCString:encoding: (page 1236)

Declared In

NSString.h

initWithCStringNoCopy:length:freeWhenDone:

Initializes the receiver, a newly allocated `NSString` object, by converting the data in a given C-string from the default C-string encoding into the Unicode character encoding. (Deprecated in iOS 2.0. Use `initWithBytesNoCopy:length:encoding:freeWhenDone:` (page 1231) instead.)

```
- (id)initWithCStringNoCopy:(char *)cString length:(NSUInteger)length
    freeWhenDone:(BOOL)flag
```

Discussion
This method converts *length* * `sizeof(char)` bytes from *cString* and doesn't stop short at a zero character. *cString* must contain data in the default C-string encoding and may not be `NULL`. The receiver becomes the owner of *cString*; if *flag* is `YES` it will free the memory when it no longer needs it, but if *flag* is `NO` it won't. Returns an initialized object, which might be different from the original receiver.

You can use this method to create an immutable string from an immutable (`const char *`) C-string buffer. If you receive a warning message, you can disregard it; its purpose is simply to warn you that the C string passed as the method's first argument may be modified. If you make certain the `freeWhenDone` argument to `initWithStringNoCopy` is `NO`, the C string passed as the method's first argument cannot be modified, so you can safely use `initWithStringNoCopy` to create an immutable string from an immutable (`const char *`) C-string buffer.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- `initWithCString:encoding:` (page 1236)

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb

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 1241) 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 iOS 2.0 and later.

See Also
`+ stringWithFormat:` (page 1203)
`- initWithFormat:locale:arguments:` (page 1241)

Related Sample Code
BonjourWeb
SpeakHere

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 1241) with `nil` as the locale.

Availability

Available in iOS 2.0 and later.

See Also

`+ stringWithFormat:` (page 1203)

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 1241) with *locale* as the locale.

Availability
Available in iOS 2.0 and later.

See Also
`+ localizedStringWithFormat:` (page 1196)

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 more information.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithFormat:arguments:` (page 1239)

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 iOS 2.0 and later.

See Also
+ `stringWithString:` (page 1204)

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 iOS 2.0 and later.

See Also
`+ stringWithUTF8String:` (page 1204)

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 iOS 2.0 and later.

See Also
`- doubleValue` (page 1218)
`- floatValue` (page 1221)
`- scanInt:` (page 1124) (`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 1243) instead.

Availability
Available in iOS 2.0 and later.

See Also
- `integerValue` (page 1243)
- `doubleValue` (page 1218)
- `floatValue` (page 1221)
- `scanInt:` (page 1124) (`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 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 516) or similar methods in `NSFileManager` for that task).

Availability
Available in iOS 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 1632).

Availability
Available in iOS 2.0 and later.

See Also
`- compare:options:range:` (page 1210)

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 iOS 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 iOS 2.0 and later.

See Also
- `lengthOfBytesUsingEncoding:` (page 1246)

Related Sample Code
CryptoExercise
MoviePlayer

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. Returns `0` if the specified encoding cannot be used to convert the receiver or if the amount of memory required for storing the results of the encoding conversion would exceed `NSIntegerMax` (page 1772).

Discussion
The result is exact and is returned in `O(n)` time.

Availability
Available in iOS 2.0 and later.

See Also
- `maximumLengthOfBytesUsingEncoding:` (page 1250)
- `length` (page 1246)

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 iOS 2.0 and later.

See Also

- `paragraphRangeForRange:` (page 1251)
- `getLineStart:end:contentsEnd:forRange:` (page 1227)
- `substringWithRange:` (page 1275)

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 iOS 2.0 and later.

See Also

- `compare:options:range:locale:` (page 1211)

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 iOS 2.0 and later.

See Also

`- compare:options:range:locale:` (page 1211)

Declared In

`NSString.h`

localizedStandardCompare:

Compares strings as sorted by the Finder.

```
- (NSComparisonResult)localizedStandardCompare:(NSString *)string
```

Parameters

string

The string to compare with the receiver.

Return Value

The result of the comparison.

Discussion

This method should be used whenever file names or other strings are presented in lists and tables where Finder-like sorting is appropriate. The exact sorting behavior of this method is different under different locales and may be changed in future releases.

Availability

Available in iOS 4.0 and later.

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 `LLONG_MAX` or `LLONG_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 iOS 2.0 and later.

See Also
- `doubleValue` (page 1218)
- `floatValue` (page 1221)
- `scanInt:` (page 1124) (`NSScanner`)

Declared In
`NSString.h`

lossyCString

Returns a representation of the receiver as a C string in the default C-string encoding, possibly losing information in converting to that encoding. (Deprecated in iOS 2.0. Use cStringUsingEncoding: (page 1215) or `dataUsingEncoding:allowLossyConversion:` (page 1216) instead.)

```
- (const char *)lossyCString
```

Discussion
This method does not raise an exception if the conversion is lossy. The returned C string will be automatically freed just as a returned object would be released; your code should copy the C string or use `getCString:` (page 1223) if it needs to store the C string outside of the autorelease context in which the C string is created.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- `cStringUsingEncoding:` (page 1215)
- `dataUsingEncoding:allowLossyConversion:` (page 1216)

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 iOS 2.0 and later.

See Also
- `capitalizedString` (page 1206)
- `uppercaseString` (page 1276)

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. Returns `0` if the amount of memory required for storing the results of the encoding conversion would exceed `NSIntegerMax` (page 1772).

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 iOS 2.0 and later.

See Also
- `lengthOfBytesUsingEncoding:` (page 1246)

- length (page 1246)

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 iOS 2.0 and later.

See Also
- getParagraphStart:end:contentsEnd:forRange: (page 1228)
- lineRangeForRange: (page 1247)

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 iOS 2.0 and later.

See Also
+ pathWithComponents: (page 1197)
- stringByStandardizingPath (page 1272)
- componentsSeparatedByString: (page 1213)

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 iOS 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 iOS 2.0 and later.

See Also
- precomposedStringWithCompatibilityMapping (page 1253)
- decomposedStringWithCanonicalMapping (page 1217)

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 iOS 2.0 and later.

See Also
- precomposedStringWithCanonicalMapping (page 1253)
- decomposedStringWithCompatibilityMapping (page 1217)

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*.

Important: Raises an `NSParseErrorException` if the receiver cannot be parsed as a property list.

Availability
Available in iOS 2.0 and later.

See Also
`- propertyListFromStringsFileFormat` (page 1254)
`+ stringWithContentsOfFile:` (page 1198)

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 iOS 2.0 and later.

See Also
`- propertyList` (page 1253)
`+ stringWithContentsOfFile:` (page 1198)

Declared In
`NSString.h`

rangeOfCharacterFromSet:

Finds and returns the range in the receiver of the first character 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 1255) with no options.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability

Available in iOS 2.0 and later.

Declared In

`NSString.h`

rangeOfCharacterFromSet:options:

Finds and returns the range in the receiver of the first character, 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 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 1256) with *mask* for the options and the entire extent of the receiver for the range.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability
Available in iOS 2.0 and later.

Declared In
`NSString.h`

rangeOfCharacterFromSet:options:range:

Finds and returns the range in the receiver of the first character 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 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}`.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `rangeOfComposedCharacterSequencesForRange:` (page 1258)

Declared In
`NSString.h`

rangeOfComposedCharacterSequencesForRange:

Returns the range in the receiver of the composed character sequences 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 that includes the composed character sequences in *range*.

Discussion
This method provides a convenient way grow a range to include all composed character sequences it overlaps.

Availability
Available in iOS 2.0 and later.

See Also
- `rangeOfComposedCharacterSequenceAtIndex:` (page 1257)

Declared In
`NSString.h`

rangeOfString:

Finds and returns the range of the first occurrence of a given string within the receiver.

```
- (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 *aString* is not found or is empty (@"").

Discussion
Invokes `rangeOfString:options:` (page 1259) with no options.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability
Available in iOS 2.0 and later.

Declared In
`NSString.h`

rangeOfString:options:

Finds and returns the range of the first occurrence of a given string within the receiver, 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 details on these options.

Return Value
An `NSRange` structure giving the location and length in the receiver of the first occurrence of *aString*, modulo the options in *mask*. Returns `{NSNotFound, 0}` if *aString* is not found or is empty (@"").

Discussion
Invokes `rangeOfString:options:range:` (page 1259) with the options specified by *mask* and the entire extent of the receiver as the range.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability
Available in iOS 2.0 and later.

Declared In
`NSString.h`

rangeOfString:options:range:

Finds and returns the range of the first occurrence of a given string, within the given range of the receiver, subject to given options.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)aRange
```

Parameters

aString

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 details on these options.

aRange

The range within the receiver for which to search for *aString*.

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 *aString* 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 *aString* is not found or is empty (@"").

Discussion

The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability

Available in iOS 2.0 and later.

Declared In

`NSString.h`

rangeOfString:options:range:locale:

Finds and returns the range of the first occurrence of a given string within a given range of the receiver, subject to given options, using the specified locale, if any.

```
- (NSRange)rangeOfString:(NSString *)aString options:(NSStringCompareOptions)mask
    range:(NSRange)searchRange locale:(NSLocale *)locale
```

Parameters

aString

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 details on these options.

aRange

The range within the receiver for which to search for *aString*.

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 *aString*. If this value is `nil`, uses the current locale.

The locale argument affects the equality checking algorithm. For example, for the Turkish locale, case-insensitive compare matches "I" to "ı" (Unicode code point U+0131, Latin Small Dotless I), not the normal "i" character.

Return Value

An `NSRange` structure giving the location and length in the receiver of *aString* 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 *aString* is not found or is empty (`@""`).

Discussion

The length of the returned range and that of *aString* may differ if equivalent composed character sequences are matched.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability

Available in iOS 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 iOS 2.0 and later.

See Also
- `fastestEncoding` (page 1220)
- `getCharacters:range:` (page 1222)

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 iOS 2.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1267)

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 iOS 2.0 and later.

See Also
`- stringByReplacingPercentEscapesUsingEncoding:` (page 1271)

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 1203).

Availability
Available in iOS 2.0 and later.

See Also
`- stringByAppendingString:` (page 1265)

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 iOS 2.0 and later.

See Also
- stringsByAppendingPaths: (page 1273)
- stringByAppendingPathExtension: (page 1264)
- stringByDeletingLastPathComponent (page 1266)

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

See Also
- stringByAppendingPathComponent: (page 1263)
- stringByDeletingPathExtension (page 1266)

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 iOS 2.0 and later.

See Also
- `stringByAppendingFormat:` (page 1263)

Related Sample Code
SimpleGestureRecognizers

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 iOS 2.0 and later.

See Also
- `stringByDeletingPathExtension` (page 1266)
- `stringByAppendingPathComponent:` (page 1263)

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 iOS 2.0 and later.

See Also
- `pathExtension` (page 1252)
- `stringByDeletingLastPathComponent` (page 1266)

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 iOS 2.0 and later.

See Also
- `stringByAbbreviatingWithTildeInPath` (page 1262)

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. The locale affects the folding logic. For example, for the Turkish locale, case-insensitive compare matches “I” to “ı” (Unicode code point U+0131, Latin Small Dotless I), not the normal “i” character.

Return Value
A string with the character folding options applied.

Discussion
Character folding operations remove distinctions between characters. For example, case folding may replace uppercase letters with their lowercase equivalents.

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
- stringByReplacingOccurrencesOfString:withString: (page 1270)
- stringByReplacingOccurrencesOfString:withString:options:range: (page 1270)
- stringByReplacingPercentEscapesUsingEncoding: (page 1271)

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 1270)with `0` options and range of the whole string.

Availability
Available in iOS 2.0 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:options:range:` (page 1270)
- `stringByReplacingCharactersInRange:withString:` (page 1269)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1271)

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 iOS 2.0 and later.

See Also
- `stringByReplacingOccurrencesOfString:withString:` (page 1270)
- `stringByReplacingCharactersInRange:withString:` (page 1269)
- `stringByReplacingPercentEscapesUsingEncoding:` (page 1271)

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 iOS 2.0 and later.

See Also
- `stringByAddingPercentEscapesUsingEncoding:` (page 1262)

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 iOS 2.0 and later.

See Also
- `stringByStandardizingPath` (page 1272)
- `stringByExpandingTildeInPath` (page 1267)

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 1271) 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 1267).
- 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 1271), 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 iOS 2.0 and later.

See Also
- `stringByExpandingTildeInPath` (page 1267)
- `stringByResolvingSymlinksInPath` (page 1271)

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 194) or `whitespaceAndNewlineCharacterSet` (page 193) to remove whitespace around strings.

Availability
Available in iOS 2.0 and later.

See Also
- `componentsSeparatedByCharactersInSet:` (page 1212)

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 1263) for an individual example.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
- `substringWithRange:` (page 1275)
- `substringToIndex:` (page 1274)

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 iOS 2.0 and later.

See Also

- `substringFromIndex:` (page 1274)
- `substringWithRange:` (page 1275)

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.

This method detects all invalid ranges (including those with negative lengths). For applications linked against Mac OS X v10.6 and later, this error causes an exception; for applications linked against earlier releases, this error causes a warning, which is displayed just once per application execution.

Availability

Available in iOS 2.0 and later.

See Also

- `substringFromIndex:` (page 1274)
- `substringToIndex:` (page 1274)

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 1249) for an example.

Availability
Available in iOS 2.0 and later.

See Also
- `capitalizedString` (page 1206)
- `lowercaseString` (page 1249)

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise

Declared In
`NSString.h`

writeToFile:atomically:

Writes the contents of the receiver to the file specified by a given path. (Deprecated in iOS 2.0. Use `writeToFile:atomically:encoding:error:` (page 1277) instead.)

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

Return Value
YES if the file is written successfully, otherwise NO.

Discussion
Writes the contents of the receiver to the file specified by *path* (overwriting any existing file at *path*). *path* is written in the default C-string encoding if possible (that is, if no information would be lost), in the Unicode encoding otherwise.

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 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.

If *path* contains a tilde (~) character, you must expand it with `stringByExpandingTildeInPath` (page 1267) before invoking this method.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
`- writeToFile:atomically:encoding:error:` (page 1277)

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 1267) 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 iOS 2.0 and later.

Declared In
NSString.h

writeToURL:atomically:

Writes the contents of the receiver to the location specified by a given URL. (Deprecated in iOS 2.0. Use writeToURL:atomically:encoding:error: (page 1278) instead.)

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

Return Value
YES if the location is written successfully, otherwise NO.

Discussion
If *atomically* is YES, the receiver is written to an auxiliary location, and then the auxiliary location is renamed to *aURL*. If *atomically* is NO, the receiver 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.

The *atomically* parameter is ignored if *aURL* is not of a type that can be accessed atomically.

Availability
Available in iOS 4.0 and later.
Deprecated in iOS 2.0.

See Also
- writeToURL:atomically:encoding:error: (page 1278)

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 iOS 2.0 and later.

Declared In
`NSString.h`

Constants

unichar

Type for Unicode characters.

```
typedef unsigned short unichar;
```

Availability
Available in iOS 2.0 and later.

Declared In
`NSString.h`

NSStringCompareOptions

Type for string comparison options.

```
typedef NSUInteger NSStringCompareOptions;
```

Discussion
See "Search and Comparison Options" (page 1279) for possible values.

Availability
Available in iOS 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,
    NSRegularExpressionSearch = 1024
};
```

Constants

`NSCaseInsensitiveSearch`

A case-insensitive search.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSLiteralSearch`

Exact character-by-character equivalence.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSBackwardsSearch`

Search from end of source string.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSAnchoredSearch`

Search is limited to start (or end, if `NSBackwardsSearch`) of source string.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSString.h`.

`NSDiacriticInsensitiveSearch`

Search ignores diacritic marks.

For example, ‘ö’ is equal to ‘o’.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWidthInsensitiveSearch`

Search ignores width differences in characters that have full-width and half-width forms, as occurs in East Asian character sets.

For example, with this option, the full-width Latin small letter 'a' (Unicode code point U+FF41) is equal to the basic Latin small letter 'a' (Unicode code point U+0061).

Available in iOS 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 iOS 2.0 and later.

  Declared in `NSString.h`.

`NSRegularExpressionSearch`
: The search string is treated as an ICU-compatible regular expression. If set, no other options can apply except `NSCaseInsensitiveSearch` and `NSAnchoredSearch`. You can use this option only with the `rangeOfString:...` methods.

  Available in iOS 3.2 and later.

  Declared in `NSString.h`.

Discussion
See Searching, Comparing, and Sorting Strings for details on the effects of these options.

Declared In
`NSString.h`

NSStringEncodingConversionOptions

Type for encoding conversion options.

```
typedef NSUInteger NSStringEncodingConversionOptions;
```

Discussion
See `NSStringEncodingConversionOptions` (page 1281) for possible values.

Availability
Available in iOS 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 iOS 2.0 and later.

  Declared in `NSString.h`.

`NSStringEncodingConversionExternalRepresentation`
Specifies an external representation (with a byte-order mark, if necessary, to indicate endianness).

Available in iOS 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 iOS 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 iOS 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 1283) for possible values.

Availability
Available in iOS 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,
    NSUTF16StringEncoding = NSUnicodeStringEncoding,
    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 iOS 2.0 and later.
Declared in `NSString.h`.

`NSNEXTSTEPStringEncoding`
8-bit ASCII encoding with NEXTSTEP extensions.
Available in iOS 2.0 and later.
Declared in `NSString.h`.

`NSJapaneseEUCStringEncoding`
8-bit EUC encoding for Japanese text.
Available in iOS 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 iOS 2.0 and later.
Declared in `NSString.h`.

`NSISOLatin1StringEncoding`
8-bit ISO Latin 1 encoding.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSSymbolStringEncoding`
8-bit Adobe Symbol encoding vector.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSNonLossyASCIIStringEncoding`
7-bit verbose ASCII to represent all Unicode characters.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSShiftJISStringEncoding`
8-bit Shift-JIS encoding for Japanese text.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSISOLatin2StringEncoding`
8-bit ISO Latin 2 encoding.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUnicodeStringEncoding`
The canonical Unicode encoding for string objects.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1251StringEncoding`
Microsoft Windows codepage 1251, encoding Cyrillic characters; equivalent to AdobeStandardCyrillic font encoding.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1252StringEncoding`
Microsoft Windows codepage 1252; equivalent to WinLatin1.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1253StringEncoding`
Microsoft Windows codepage 1253, encoding Greek characters.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1254StringEncoding`
Microsoft Windows codepage 1254, encoding Turkish characters.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSWindowsCP1250StringEncoding`
Microsoft Windows codepage 1250; equivalent to WinLatin2.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSISO2022JPStringEncoding`
ISO 2022 Japanese encoding for email.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSMacOSRomanStringEncoding`
Classic Macintosh Roman encoding.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF16StringEncoding`
An alias for `NSUnicodeStringEncoding`.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF16BigEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF16LittleEndianStringEncoding`
`NSUTF16StringEncoding` encoding with explicit endianness specified.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF32StringEncoding`
32-bit UTF encoding.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF32BigEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in iOS 2.0 and later.

Declared in `NSString.h`.

`NSUTF32LittleEndianStringEncoding`
`NSUTF32StringEncoding` encoding with explicit endianness specified.

Available in iOS 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 *String 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.

String Enumeration Options

Constants to specify kinds of substrings and styles of enumeration.

```
typedef NSUInteger NSStringEnumerationOptions;
enum {
NSStringEnumerationByLines = 0,
NSStringEnumerationByParagraphs = 1,
NSStringEnumerationByComposedCharacterSequences = 2,
NSStringEnumerationByWords = 3,
NSStringEnumerationBySentences = 4,
NSStringEnumerationReverse = 1UL << 8,
NSStringEnumerationSubstringNotRequired = 1UL << 9,
NSStringEnumerationLocalized = 1UL << 10
};
```

Constants

`NSStringEnumerationByLines`
Enumerates by lines. Equivalent to `lineRangeForRange:` (page 1247).

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationByParagraphs`
Enumerates by paragraphs. Equivalent to `paragraphRangeForRange:` (page 1251).

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationByComposedCharacterSequences`
Enumerates by composed character sequences. Equivalent to `rangeOfComposedCharacterSequencesForRange:` (page 1258).

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationByWords`
Enumerates by words.

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationBySentences`
Enumerates by sentences.

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationReverse`
Causes enumeration to occur from the end of the specified range to the start.

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationSubstringNotRequired`
A way to indicate that the block does not need substring, in which case nil will be passed. This is simply a performance shortcut.

Available in iOS 4.0 and later.

Declared in `NSString.h`.

`NSStringEnumerationLocalized`

Causes the enumeration to occur using user's default locale. This does not make a difference in line, paragraph, or composed character sequence enumeration, but it may for words or sentences.

Available in iOS 4.0 and later.

Declared in `NSString.h`.

Discussion

These options are used with the `enumerateSubstringsInRange:options:usingBlock:` (page 1219) method. Pass in one `NSStringEnumerationBy...` option and combine with any of the remaining enumeration style constants using the C bitwise `OR` operator.

NSTextCheckingResult Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSTextCheckingResult.h
Companion guide	Spell Checking

Overview

`NSTextCheckingResult` is a class used to describe items located by text checking. Each of these objects represents something that has been found during checking–a misspelled word, a sentence with grammatical issues, a detected URL, a straight quote to be replaced with curly quotes, and so forth.

Instances of `NSTextCheckingResult` are returned by the `NSSpellChecker` to describe the results of spelling, grammar, or substitution actions. They are also returned by the `NSRegularExpression` class and the `NSDataDetector` class results to indicate the discovery of content. In those cases what is found may be a match for a regular expression or a date, address, phone number, etc.

Tasks

Text Checking Type Range and Type

`range` (page 1294) *property*
Returns the range of the result that the receiver represents. (read-only)

`resultType` (page 1295) *property*
Returns the text checking result type that the receiver represents. (read-only)

`numberOfRanges` (page 1293) *property*
Returns the number of ranges. (read-only)

`- rangeAtIndex:` (page 1303)
Returns the result type that the range represents.

Text Checking Results for Text Replacement

`+ replacementCheckingResultWithRange:replacementString:` (page 1302)
Creates and returns a text checking result with the specified replacement string.

`replacementString` (page 1295) *property*
A replacement string from one of a number of replacement checking results.. (read-only)

Text Checking Results for Regular Expressions

`+ regularExpressionCheckingResultWithRanges:count:regularExpression:` (page 1301)
Creates and returns a type checking result with the specified regular expression data.

`regularExpression` (page 1295) *property*
The regular expression of a type checking result. (read-only)

Text Checking Result Components

`components` (page 1292) *property*
A dictionary containing the components of a type checking result. (read-only)

Text Checking Results for URLs

`+ linkCheckingResultWithRange:URL:` (page 1299)
Creates and returns a text checking result with the specified URL.

`URL` (page 1296) *property*
The URL of a type checking result. (read-only)

Text Checking Results for Addresses

`+ addressCheckingResultWithRange:components:` (page 1296)
Creates and returns a text checking result with the specified address components.

`addressComponents` (page 1292) *property*
The address dictionary of a type checking result. (read-only)

Text Checking Results for Transit Information

`+ transitInformationCheckingResultWithRange:components:` (page 1303)
Creates and returns a text checking result with the specified transit information.

Text Checking Results for Phone Numbers

`+ phoneNumberCheckingResultWithRange:phoneNumber:` (page 1300)
Creates and returns a text checking result with the specified phone number.

`phoneNumber` (page 1294) *property*
The phone number of a type checking result. (read-only)

Text Checking Results for Dates and Times

`+ dateCheckingResultWithRange:date:` (page 1298)
Creates and returns a text checking result with the specified date.

`+ dateCheckingResultWithRange:date:timeZone:duration:` (page 1298)
Creates and returns a text checking result with the specified date, time zone, and duration..

`date` (page 1292) *property*
The date component of a type checking result. (read-only)

`duration` (page 1293) *property*
The duration component of a type checking result. (read-only)

`timeZone` (page 1296) *property*
The time zone component of a type checking result. (read-only)

Text Checking Results for Typography

`+ dashCheckingResultWithRange:replacementString:` (page 1297)
Creates and returns a text checking result with the specified dash corrected replacement string.

`+ quoteCheckingResultWithRange:replacementString:` (page 1301)
Creates and returns a text checking result with the specified quote balanced replacement string.

Text Checking Results for Spelling

`+ spellCheckingResultWithRange:` (page 1302)
Creates and returns a text checking result with the range of a misspelled word.

`+ correctionCheckingResultWithRange:replacementString:` (page 1297)
Creates and returns a text checking result after detecting a possible correction.

Text Checking Results for Orthography

`+ orthographyCheckingResultWithRange:orthography:` (page 1300)
Creates and returns a text checking result with the specified orthography.

`orthography` (page 1294) *property*
The detected orthography of a type checking result. (read-only)

Text Checking Results for Grammar

`+ grammarCheckingResultWithRange:details:` (page 1299)
Creates and returns a text checking result with the specified array of grammatical errors.

`grammarDetails` (page 1293) *property*
The details of a located grammatical type checking result. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

addressComponents

The address dictionary of a type checking result. (read-only)

```
@property(readonly) NSDictionary *addressComponents
```

Discussion
The dictionary keys are described in "Keys for Address Components" (page 1304).

Availability
Available in iOS 4.0 and later.

See Also
`+ addressCheckingResultWithRange:components:` (page 1296)

Declared In
`NSTextCheckingResult.h`

components

A dictionary containing the components of a type checking result. (read-only)

```
@property(readonly) NSDictionary *components
```

Discussion
Currently used by the transit checking result. The supported keys are located in "Keys for Transit Components" (page 1304).

Availability
Available in iOS 4.0 and later.

See Also
`+ transitInformationCheckingResultWithRange:components:` (page 1303)

Declared In
`NSTextCheckingResult.h`

date

The date component of a type checking result. (read-only)

```
@property(readonly) NSDate *date
```

Availability
Available in iOS 4.0 and later.

See Also
+ dateCheckingResultWithRange:date: (page 1298)
+ dateCheckingResultWithRange:date:timeZone:duration: (page 1298)

Declared In
NSTextCheckingResult.h

duration

The duration component of a type checking result. (read-only)

```
@property(readonly) NSTimeInterval duration
```

Availability
Available in iOS 4.0 and later.

See Also
+ dateCheckingResultWithRange:date: (page 1298)
+ dateCheckingResultWithRange:date:timeZone:duration: (page 1298)

Declared In
NSTextCheckingResult.h

grammarDetails

The details of a located grammatical type checking result. (read-only)

```
@property(readonly) NSArray *grammarDetails
```

Discussion
This array of strings is suitable for presenting to the user.

Availability
Available in iOS 4.0 and later.

See Also
+ grammarCheckingResultWithRange:details: (page 1299)

Declared In
NSTextCheckingResult.h

numberOfRanges

Returns the number of ranges. (read-only)

```
@property(readonly) NSUInteger numberOfRanges
```

Discussion
A result must have at least one range, but may optionally have more (for example, to represent regular expression capture groups).

Passing `rangeAtIndex:` (page 1303) the value `0` always returns the value of the the `range` (page 1294) property. Additional ranges, if any, will have indexes from `1` to `numberOfRanges-1`.

Availability
Available in iOS 4.0 and later.

Declared In
`NSTextCheckingResult.h`

orthography

The detected orthography of a type checking result. (read-only)

```
@property(readonly) NSOrthography *orthography
```

Availability
Available in iOS 4.0 and later.

See Also
`+ orthographyCheckingResultWithRange:orthography:` (page 1300)

Declared In
`NSTextCheckingResult.h`

phoneNumber

The phone number of a type checking result. (read-only)

```
@property(readonly) NSString *phoneNumber
```

Availability
Available in iOS 4.0 and later.

See Also
`+ phoneNumberCheckingResultWithRange:phoneNumber:` (page 1300)

Declared In
`NSTextCheckingResult.h`

range

Returns the range of the result that the receiver represents. (read-only)

```
@property(readonly) NSRange range
```

Discussion
This property will be present for all returned `NSTextCheckingResult` instances.

Availability
Available in iOS 4.0 and later.

See Also
`@property resultType` (page 1295)

Declared In
`NSTextCheckingResult.h`

regularExpression

The regular expression of a type checking result. (read-only)

```
@property(readonly) NSRegularExpression *regularExpression
```

Availability
Available in iOS 4.0 and later.

See Also
`+ regularExpressionCheckingResultWithRanges:count:regularExpression:` (page 1301)

Declared In
`NSTextCheckingResult.h`

replacementString

A replacement string from one of a number of replacement checking results.. (read-only)

```
@property(readonly) NSString *replacementString
```

Availability
Available in iOS 4.0 and later.

See Also
`+ replacementCheckingResultWithRange:replacementString:` (page 1302)

Declared In
`NSTextCheckingResult.h`

resultType

Returns the text checking result type that the receiver represents. (read-only)

```
@property(readonly) NSTextCheckingType resultType
```

Discussion
The possible result types for the built in checking capabilities are described in "NSTextCheckingType" (page 1306).

This property will be present for all returned `NSTextCheckingResult` instances.

Availability
Available in iOS 4.0 and later.

Declared In
`NSTextCheckingResult.h`

timeZone

The time zone component of a type checking result. (read-only)

```
@property(readonly) NSTimeZone *timeZone
```

Availability
Available in iOS 4.0 and later.

See Also
`+ dateCheckingResultWithRange:date:` (page 1298)
`+ dateCheckingResultWithRange:date:timeZone:duration:` (page 1298)

Declared In
`NSTextCheckingResult.h`

URL

The URL of a type checking result. (read-only)

```
@property(readonly) NSURL *URL
```

Availability
Available in iOS 4.0 and later.

See Also
`+ linkCheckingResultWithRange:URL:` (page 1299)

Declared In
`NSTextCheckingResult.h`

Class Methods

addressCheckingResultWithRange:components:

Creates and returns a text checking result with the specified address components.

```
+ (NSTextCheckingResult *)addressCheckingResultWithRange:(NSRange)range
    components:(NSDictionary *)components
```

Parameters

range

The range of the detected result.

components

A dictionary containing the address components. The dictionary keys are described in "Keys for Address Components" (page 1304).

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeAddress` (page 1306).

Availability

Available in iOS 4.0 and later.

See Also

`@property addressComponents` (page 1292)

Declared In

`NSTextCheckingResult.h`

correctionCheckingResultWithRange:replacementString:

Creates and returns a text checking result after detecting a possible correction.

```
+ (NSTextCheckingResult *)correctionCheckingResultWithRange:(NSRange)range
    replacementString:(NSString *)replacementString
```

Parameters

range

The range of the detected result.

replacementString

The suggested replacement string.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeSpelling` (page 1306).

Availability

Available in iOS 4.0 and later.

See Also

`@property replacementString` (page 1295)

Declared In

`NSTextCheckingResult.h`

dashCheckingResultWithRange:replacementString:

Creates and returns a text checking result with the specified dash corrected replacement string.

```
+ (NSTextCheckingResult *)dashCheckingResultWithRange:(NSRange)range
    replacementString:(NSString *)replacementString
```

Parameters

range

The range of the detected result.

replacementString

The replacement string.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeDash` (page 1307).

Availability

Available in iOS 4.0 and later.

Declared In

`NSTextCheckingResult.h`

dateCheckingResultWithRange:date:

Creates and returns a text checking result with the specified date.

```
+ (NSTextCheckingResult *)dateCheckingResultWithRange:(NSRange)range date:(NSDate
    *)date
```

Parameters

range

The range of the detected result.

date

The detected date.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeDate` (page 1306).

Availability

Available in iOS 4.0 and later.

Declared In

`NSTextCheckingResult.h`

dateCheckingResultWithRange:date:timeZone:duration:

Creates and returns a text checking result with the specified date, time zone, and duration..

```
+ (NSTextCheckingResult *)dateCheckingResultWithRange:(NSRange)range date:(NSDate
    *)date timeZone:(NSTimeZone *)timeZone duration:(NSTimeInterval)duration
```

Parameters

range

The range of the detected result.

date
The detected date.

timeZone
The detected time zone.

duration
The detected duration.

Return Value
Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeDate` (page 1306).

Availability
Available in iOS 4.0 and later.

See Also
`@property date` (page 1292)
`@property duration` (page 1293)
`@property timeZone` (page 1296)

Declared In
`NSTextCheckingResult.h`

grammarCheckingResultWithRange:details:

Creates and returns a text checking result with the specified array of grammatical errors.

```
+ (NSTextCheckingResult *)grammarCheckingResultWithRange:(NSRange)range
    details:(NSArray *)details
```

Parameters

range
The range of the detected result.

details
An array of details regarding the grammatical errors. This array of strings is suitable for presenting to the user.

Return Value
Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeGrammar` (page 1306).

Availability
Available in iOS 4.0 and later.

See Also
`@property grammarDetails` (page 1293)

Declared In
`NSTextCheckingResult.h`

linkCheckingResultWithRange:URL:

Creates and returns a text checking result with the specified URL.

```
+ (NSTextCheckingResult *)linkCheckingResultWithRange:(NSRange)range URL:(NSURL
    *)url
```

Parameters

range

The range of the detected result.

url

The URL.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeLink` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`@property URL` (page 1296)

Declared In

`NSTextCheckingResult.h`

orthographyCheckingResultWithRange:orthography:

Creates and returns a text checking result with the specified orthography.

```
+ (NSTextCheckingResult *)orthographyCheckingResultWithRange:(NSRange)range
    orthography:(NSOrthography *)orthography
```

Parameters

range

The range of the detected result.

orthography

An orthography object that describes the script.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeOrthography` (page 1306).

Availability

Available in iOS 4.0 and later.

See Also

`@property orthography` (page 1294)

Declared In

`NSTextCheckingResult.h`

phoneNumberCheckingResultWithRange:phoneNumber:

Creates and returns a text checking result with the specified phone number.

```
+ (NSTextCheckingResult *)phoneNumberCheckingResultWithRange:(NSRange)range
    phoneNumber:(NSString *)phoneNumber
```

Parameters

range

The range of the detected result.

phoneNumber

The phone number.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypePhoneNumber` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`@property phoneNumber` (page 1294)

Declared In

`NSTextCheckingResult.h`

quoteCheckingResultWithRange:replacementString:

Creates and returns a text checking result with the specified quote balanced replacement string.

```
+ (NSTextCheckingResult *)quoteCheckingResultWithRange:(NSRange)range
    replacementString:(NSString *)replacementString
```

Parameters

range

The range of the detected result.

replacementString

The replacement string.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeQuote` (page 1307).

Availability

Available in iOS 4.0 and later.

Declared In

`NSTextCheckingResult.h`

regularExpressionCheckingResultWithRanges:count:regularExpression:

Creates and returns a type checking result with the specified regular expression data.

```
+ (NSTextCheckingResult
    *)regularExpressionCheckingResultWithRanges:(NSRangePointer)ranges
    count:(NSUInteger)count regularExpression:(NSRegularExpression
    *)regularExpression
```

Parameters

ranges

A C array of ranges, which must have at least one element, the first element represents the overall range.

count

The number of items in the *ranges* array.

regularExpression

The regular expression.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeRegularExpression` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`@property regularExpression` (page 1295)

Declared In

`NSTextCheckingResult.h`

replacementCheckingResultWithRange:replacementString:

Creates and returns a text checking result with the specified replacement string.

```
+ (NSTextCheckingResult *)replacementCheckingResultWithRange:(NSRange)range
    replacementString:(NSString *)replacementString
```

Parameters

range

The range of the detected result.

replacementString

The replacement string.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeReplacement` (page 1307).

Availability

Available in iOS 4.0 and later.

See Also

`@property replacementString` (page 1295)

Declared In

`NSTextCheckingResult.h`

spellCheckingResultWithRange:

Creates and returns a text checking result with the range of a misspelled word.

```
+ (NSTextCheckingResult *)spellCheckingResultWithRange:(NSRange)range
```

Parameters

range

The range of the detected result.

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeSpelling` (page 1306).

Availability

Available in iOS 4.0 and later.

Declared In

`NSTextCheckingResult.h`

transitInformationCheckingResultWithRange:components:

Creates and returns a text checking result with the specified transit information.

```
+ (NSTextCheckingResult *)transitInformationCheckingResultWithRange:(NSRange)range
    components:(NSDictionary *)components
```

Parameters

range

The range of the detected result.

components

A dictionary containing the transit components. The currently supported keys are `NSTextCheckingAirlineKey` (page 1304) and `NSTextCheckingFlightKey` (page 1304).

Return Value

Returns an `NSTextCheckingResult` with the specified `range` (page 1294) and a `resultType` (page 1295) of `NSTextCheckingTypeTransitInformation` (page 1307).

Availability

Available in iOS 4.0 and later.

Declared In

`NSTextCheckingResult.h`

Instance Methods

rangeAtIndex:

Returns the result type that the range represents.

```
- (NSRange)rangeAtIndex:(NSUInteger)idx
```

Parameters

idx

The index of the result.

Return Value
The range of the result.

Discussion
A result must have at least one range, but may optionally have more (for example, to represent regular expression capture groups).

Passing `rangeAtIndex:` (page 1303) the value `0` always returns the value of the the `range` (page 1294) property. Additional ranges, if any, will have indexes from `1` to `numberOfRanges-1`.

Availability
Available in iOS 4.0 and later.

Declared In
`NSTextCheckingResult.h`

Constants

Keys for Transit Components

The following constants identify the possible keys returned in the components dictionary.

```
NSString * const NSTextCheckingAirlineKey;
NSString * const NSTextCheckingFlightKey;
```

Constants

`NSTextCheckingAirlineKey`
A key that corresponds to the airline of a transit result.
Available in iOS 4.0 and later.
Declared in `NSTextCheckingResult.h`.

`NSTextCheckingFlightKey`
A key that corresponds to the flight component of a transit result.
Available in iOS 4.0 and later.
Declared in `NSTextCheckingResult.h`.

Keys for Address Components

The following constants identify the possible keys returned in the `addressComponents` (page 1292) dictionary.

```
NSString * const NSTextCheckingNameKey;
NSString * const NSTextCheckingJobTitleKey;
NSString * const NSTextCheckingOrganizationKey;
NSString * const NSTextCheckingStreetKey;
NSString * const NSTextCheckingCityKey;
NSString * const NSTextCheckingStateKey;
NSString * const NSTextCheckingZIPKey;
NSString * const NSTextCheckingCountryKey;
NSString * const NSTextCheckingPhoneKey;
```

Constants

`NSTextCheckingNameKey`

A key that corresponds to the name component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingJobTitleKey`

A key that corresponds to the job component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingOrganizationKey`

A key that corresponds to the organization component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingStreetKey`

A key that corresponds to the street address component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingCityKey`

A key that corresponds to the city component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingStateKey`

A key that corresponds to the state or province component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingZIPKey`

A key that corresponds to the zip code or postal code component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingCountryKey`

A key that corresponds to the country component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingPhoneKey`
A key that corresponds to the phone number component of the address.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

NSTextCheckingType

These constants specify the type of checking the methods should do. They are returned by `resultType` (page 1295).

```
enum {
    NSTextCheckingTypeOrthography        = 1ULL << 0,
    NSTextCheckingTypeSpelling           = 1ULL << 1,
    NSTextCheckingTypeGrammar            = 1ULL << 2,
    NSTextCheckingTypeDate               = 1ULL << 3,
    NSTextCheckingTypeAddress            = 1ULL << 4,
    NSTextCheckingTypeLink               = 1ULL << 5,
    NSTextCheckingTypeQuote              = 1ULL << 6,
    NSTextCheckingTypeDash               = 1ULL << 7,
    NSTextCheckingTypeReplacement        = 1ULL << 8,
    NSTextCheckingTypeCorrection         = 1ULL << 9,
    NSTextCheckingTypeRegularExpression  = 1ULL << 10
    NSTextCheckingTypePhoneNumber        = 1ULL << 11,
    NSTextCheckingTypeTransitInformation = 1ULL << 12
};
typedef uint64_t NSTextCheckingType;
```

Constants

`NSTextCheckingTypeOrthography`
Attempts to identify the language

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeSpelling`
Checks spelling.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeGrammar`
Checks grammar.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeDate`
Attempts to locate dates.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeAddress`
Attempts to locate addresses.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeLink`
: Attempts to locate URL links.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeQuote`
: Replaces quotes with smart quotes..

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeDash`
: Replaces dashes with em-dashes.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeReplacement`
: Replaces characters such as (c) with the appropriate symbol (in this case ©).

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeCorrection`
: Performs autocorrection on misspelled words.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeRegularExpression`
: Matches a regular expression.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypePhoneNumber`
: Matches a phone number.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

`NSTextCheckingTypeTransitInformation`
: Matches a transit information, for example, flight information.

  Available in iOS 4.0 and later.

  Declared in `NSTextCheckingResult.h`.

NSTextCheckingTypes

Defines the types of checking that are available. These values can be combined using the C-bitwise OR operator. The system supports its own internal types, and the user can extend those types by subclassing `NSTextCheckingResult` and adding their own custom types.

```
enum {
    NSTextCheckingAllSystemTypes    = 0xffffffffULL,
    NSTextCheckingAllCustomTypes    = 0xffffffffULL << 32,purposes
    NSTextCheckingAllTypes          = (NSTextCheckingAllSystemTypes |
NSTextCheckingAllCustomTypes)
};
typedef uint64_t NSTextCheckingTypes;
```

Constants

`NSTextCheckingAllSystemTypes`

Checking types supported by the system. The first 32 types are reserved.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingAllCustomTypes`

Checking types that can be used by clients.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

`NSTextCheckingAllTypes`

All possible checking types, both system and user supported.

Available in iOS 4.0 and later.

Declared in `NSTextCheckingResult.h`.

NSThread Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSThread.h
Companion guide	Threading Programming Guide

Overview

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 1313) 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 1322) 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 1317) 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 1317)
  Returns an initialized `NSThread` object.
- `initWithTarget:selector:object:` (page 1318)
  Returns an `NSThread` object initialized with the given arguments.

Starting a Thread

+ `detachNewThreadSelector:toTarget:withObject:` (page 1313)
  Detaches a new thread and uses the specified selector as the thread entry point.
- `start` (page 1322)
  Starts the receiver.
- `main` (page 1320)
  The main entry point routine for the thread.

Stopping a Thread

+ `sleepUntilDate:` (page 1316)
  Blocks the current thread until the time specified.
+ `sleepForTimeInterval:` (page 1315)
  Sleeps the thread for a given time interval.
+ `exit` (page 1313)
  Terminates the current thread.
- `cancel` (page 1317)
  Changes the cancelled state of the receiver to indicate that it should exit.

Determining the Thread's Execution State

- `isExecuting` (page 1319)
  Returns a Boolean value that indicates whether the receiver is executing.
- `isFinished` (page 1319)
  Returns a Boolean value that indicates whether the receiver has finished execution.
- `isCancelled` (page 1318)
  Returns a Boolean value that indicates whether the receiver is cancelled.

Working with the Main Thread

`+ isMainThread` (page 1314)
Returns a Boolean value that indicates whether the current thread is the main thread.

`- isMainThread` (page 1319)
Returns a Boolean value that indicates whether the receiver is the main thread.

`+ mainThread` (page 1315)
Returns the `NSThread` object representing the main thread.

Querying the Environment

`+ isMultiThreaded` (page 1314)
Returns whether the application is multithreaded.

`+ currentThread` (page 1312)
Returns the thread object representing the current thread of execution.

`+ callStackReturnAddresses` (page 1312)
Returns an array containing the call stack return addresses.

`+ callStackSymbols` (page 1312)
Returns an array containing the call stack symbols.

Working with Thread Properties

`- threadDictionary` (page 1322)
Returns the thread object's dictionary.

`- name` (page 1320)
Returns the name of the receiver.

`- setName:` (page 1321)
Sets the name of the receiver.

`- stackSize` (page 1322)
Returns the stack size of the receiver.

`- setStackSize:` (page 1321)
Sets the stack size of the receiver.

Working with Thread Priorities

`+ threadPriority` (page 1316)
Returns the current thread's priority.

`- threadPriority` (page 1323)
Returns the reciever's priority

`+ setThreadPriority:` (page 1315)
Sets the current thread's priority.

`- setThreadPriority:` (page 1321)
Sets the receiver'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 iOS 2.0 and later.

Declared In
`NSThread.h`

callStackSymbols

Returns an array containing the call stack symbols.

```
+ (NSArray *)callStackSymbols
```

Return Value
An array containing the call stack symbols.

Discussion
This method returns an array of strings describing the call stack backtrace of the current thread at the moment this method was called. The format of each string is non-negotiable and is determined by the `backtrace_symbols()` API

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

See Also
`+ detachNewThreadSelector:toTarget:withObject:` (page 1313)

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 1313) 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 1324) with object nil to the default notification center.

Availability
Available in iOS 2.0 and later.

See Also
+ currentThread (page 1312)
+ isMultiThreaded (page 1314)
- start (page 1322)

Declared In
NSThread.h

exit

Terminates the current thread.

```
+ (void)exit
```

Discussion
This method uses the `currentThread` (page 1312) class method to access the current thread. Before exiting the thread, this method posts the `NSThreadWillExitNotification` (page 1324) with the thread being exited to the default notification center. Because notifications are delivered synchronously, all observers of `NSThreadWillExitNotification` (page 1324) 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 iOS 2.0 and later.

See Also
`+ currentThread` (page 1312)
`+ sleepUntilDate:` (page 1316)

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 iOS 2.0 and later.

See Also
`+ mainThread` (page 1315)

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 1313) or `start` (page 1322). 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 iOS 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 iOS 2.0 and later.

See Also
- isMainThread (page 1319)

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 iOS 2.0 and later.

See Also
+ threadPriority (page 1316)

Declared In
NSThread.h

sleepForTimeInterval:

Sleeps the thread 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 iOS 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 iOS 2.0 and later.

See Also

+ currentThread (page 1312)

+ exit (page 1313)

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 iOS 2.0 and later.

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 iOS 2.0 and later.

See Also
`- isCancelled` (page 1318)

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 iOS 2.0 and later.

See Also
`- initWithTarget:selector:object:` (page 1318)
`- start` (page 1322)

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 iOS 2.0 and later.

See Also
- init (page 1317)
- start (page 1322)

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 iOS 2.0 and later.

See Also
- `cancel` (page 1317)
- `isExecuting` (page 1319)
- `isFinished` (page 1319)

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 iOS 2.0 and later.

See Also
- `isCancelled` (page 1318)
- `isFinished` (page 1319)

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 iOS 2.0 and later.

See Also
- `isCancelled` (page 1318)
- `isExecuting` (page 1319)

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 iOS 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 iOS 2.0 and later.

See Also
`- start` (page 1322)

Declared In
`NSThread.h`

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value
The name of the receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- setName:` (page 1321)

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 iOS 2.0 and later.

See Also
- name (page 1320)

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 in bytes and 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 1322) method), but it does not affect the actual number of pages set aside for the thread.

Availability
Available in iOS 2.0 and later.

See Also
- stackSize (page 1322)

Declared In
NSThread.h

setThreadPriority:

Sets the receiver's priority.

```
- (void)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.

Discussion
The priorities in this range are mapped to the operating system's priority values.

Availability
Available in iOS 4.0 and later.

Declared In
NSThread.h

stackSize

Returns the stack size of the receiver.

```
- (NSUInteger)stackSize
```

Return Value
The stack size of the receiver, in bytes.

Availability
Available in iOS 2.0 and later.

See Also
- setStackSize: (page 1321)

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 1324) with object nil to the default notification center.

Availability
Available in iOS 2.0 and later.

See Also
- init (page 1317)
- initWithTarget:selector:object: (page 1318)
- main (page 1320)

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 iOS 2.0 and later.

Declared In
`NSThread.h`

threadPriority

Returns the reciever's priority

```
- (double)threadPriority
```

Return Value
The 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 iOS 4.0 and later.

Declared In
`NSThread.h`

Notifications

NSDidBecomeSingleThreadedNotification

Not implemented.

Availability
Available in iOS 2.0 and later.

Declared In
`NSThread.h`

NSThreadWillExitNotification

An `NSThread` object posts this notification when it receives the `exit` (page 1313) 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 iOS 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 1313) or the `start` (page 1322) 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSTimer.h
Companion guides	Timer Programming Topics Threading Programming Guide
Related sample code	AddMusic BonjourWeb GKRocket ScrollViewSuite WiTap

Overview

You use the `NSTimer` class to create 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 target 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 *Threading Programming Guide*. Note in particular that run loops retain their timers, so you can release a timer after you have added it to a run loop.

A timer is not a real-time mechanism; it fires only when one of the run loop modes to which the timer has been added is running and able to check if the timer's firing time has passed. Because of the various input sources a typical run loop manages, the effective resolution of the time interval for a timer is limited to on the order of 50-100 milliseconds. If a timer's firing time occurs during a long callout or while the run loop is in a mode that is not monitoring the timer, the timer does not fire until the next time the run loop checks the timer. Therefore, the actual time at which the timer fires potentially can be a significant period of time after the scheduled firing time.

`NSTimer` objects are "toll-free bridged" with their Core Foundation counterparts, the `CFRunLoopTimerRef` type. 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. For more information about Core Foundation timers, see *CFRunLoopTimer Reference*.

Repeating Versus Non-Repeating Timers

You specify whether a timer is repeating or non-repeating at creation time. A non-repeating timer fires once and then invalidates itself automatically, thereby preventing the timer from firing again. By contrast, a repeating timer fires and then reschedules itself on the same run loop.

A repeating timer always schedules itself based on the scheduled firing time, as opposed to the actual firing time. For example, if a timer is scheduled to fire at a particular time and every 5 seconds after that, the scheduled firing time will always fall on the original 5 second time intervals, even if the actual firing time gets delayed. If the firing time is delayed so far that it passes one or more of the scheduled firing times, the timer is fired only once for that time period; the timer is then rescheduled, after firing, for the next scheduled firing time in the future.

Scheduling Timers in Run Loops

A timer object can be registered in only one run loop at a time, although it can be added to multiple run loop modes within that run loop. There are three ways to create a timer:

- Use the `scheduledTimerWithTimeInterval:invocation:repeats:` (page 1328) or `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1328) class method to create the timer and schedule it on the current run loop in the default mode.
- Use the `timerWithTimeInterval:invocation:repeats:` (page 1329) or `timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1330) class method to create the timer object without scheduling it on a run loop. (After creating it, you must add the timer to a run loop manually by calling the `addTimer:forMode:` (page 1107) method of the corresponding `NSRunLoop` object.)
- Allocate the timer and initialize it using the `initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1331) method. (After creating it, you must add the timer to a run loop manually by calling the `addTimer:forMode:` (page 1107) method of the corresponding `NSRunLoop` object.)

Once scheduled on a run loop, the timer fires at the specified interval until it is invalidated. A non-repeating timer invalidates itself immediately after it fires. However, for a repeating timer, you must invalidate the timer object yourself by calling its `invalidate` (page 1332) method. Calling this method requests the removal of the timer from the current run loop; as a result, you should always call the `invalidate` (page 1332) method from the same thread on which the timer was installed. Invalidating the timer immediately disables it so that it no longer affects the run loop. The run loop then removes and releases the timer, either just before the `invalidate` (page 1332) method returns or at some later point. Once invalidated, timer objects cannot be reused.

Tasks

Creating a Timer

`+ scheduledTimerWithTimeInterval:invocation:repeats:` (page 1328)
Creates and returns a new `NSTimer` object and schedules it on the current run loop in the default mode.

`+ scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1328)
Creates and returns a new `NSTimer` object and schedules it on the current run loop in the default mode.

`+ timerWithTimeInterval:invocation:repeats:` (page 1329)
Creates and returns a new `NSTimer` object initialized with the specified invocation object.

`+ timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1330)
Creates and returns a new `NSTimer` object initialized with the specified object and selector.

`- initWithFireDate:interval:target:selector:userInfo:repeats:` (page 1331)
Initializes a new `NSTimer` object using the specified object and selector.

Firing a Timer

`- fire` (page 1331)
Causes the receiver's message to be sent to its target.

Stopping a Timer

`- invalidate` (page 1332)
Stops the receiver from ever firing again and requests its removal from its run loop.

Information About a Timer

`- isValid` (page 1333)
Returns a Boolean value that indicates whether the receiver is currently valid.

`- fireDate` (page 1331)
Returns the date at which the receiver will fire.

`- setFireDate:` (page 1333)
Resets the firing time of the receiver to the specified date.

`- timeInterval` (page 1334)
Returns the receiver's time interval.

`- userInfo` (page 1334)
Returns the receiver's *`userInfo`* object.

Class Methods

scheduledTimerWithTimeInterval:invocation:repeats:

Creates and returns a new `NSTimer` object and schedules it on the current run loop in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    invocation:(NSInvocation *)invocation  repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer. If *seconds* is less than or equal to `0.0`, this method chooses the nonnegative value of 0.1 milliseconds instead.

invocation

The invocation to use when the timer fires. The timer instructs the invocation object to retain its arguments.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
After *seconds* seconds have elapsed, the timer fires, invoking *invocation*.

Availability
Available in iOS 2.0 and later.

Declared In
`NSTimer.h`

scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:

Creates and returns a new `NSTimer` object and schedules it on the current run loop in the default mode.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
    target:(id)target selector:(SEL)aSelector  userInfo:(id)userInfo
    repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer. If *seconds* is less than or equal to `0.0`, this method chooses the nonnegative value of 0.1 milliseconds instead.

target

The object to which to send the message specified by *aSelector* when the timer fires. The target object is retained by the timer and released when the timer is invalidated.

aSelector

The message to send to *target* when the timer fires. The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo

The user info for the timer. The object you specify is retained by the timer and released when the timer is invalidated. This parameter may be `nil`.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
After *seconds* seconds have elapsed, the timer fires, sending the message *aSelector* to *target*.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
GKRocket
GKTank
ScrollViewSuite
WiTap

Declared In
`NSTimer.h`

timerWithTimeInterval:invocation:repeats:

Creates and returns a new `NSTimer` object initialized with the specified invocation object.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds  invocation:(NSInvocation
    *)invocation  repeats:(BOOL)repeats
```

Parameters

seconds

The number of seconds between firings of the timer. If *seconds* is less than or equal to `0.0`, this method chooses the nonnegative value of 0.1 milliseconds instead

invocation

The invocation to use when the timer fires. The timer instructs the invocation object to retain its arguments.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
You must add the new timer to a run loop, using `addTimer:forMode:` (page 1107). Then, after *seconds* have elapsed, the timer fires, invoking *invocation*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability
Available in iOS 2.0 and later.

Declared In
`NSTimer.h`

timerWithTimeInterval:target:selector:userInfo:repeats:

Creates and returns a new `NSTimer` object initialized with the specified object and selector.

```
+ (NSTimer *)timerWithTimeInterval:(NSTimeInterval)seconds  target:(id)target
    selector:(SEL)aSelector  userInfo:(id)userInfo  repeats:(BOOL)repeats
```

Parameters

seconds
The number of seconds between firings of the timer. If *seconds* is less than or equal to `0.0`, this method chooses the nonnegative value of 0.1 milliseconds instead.

target
The object to which to send the message specified by *aSelector* when the timer fires. The target object is retained by the timer and released when the timer is invalidated.

aSelector
The message to send to *target* when the timer fires. The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo
Custom user info for the timer. The object you specify is retained by the timer and released when the timer is invalidated. This parameter may be `nil`.

repeats
If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value
A new `NSTimer` object, configured according to the specified parameters.

Discussion
You must add the new timer to a run loop, using `addTimer:forMode:` (page 1107). Then, after *seconds* seconds have elapsed, the timer fires, sending the message *aSelector* to *target*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability
Available in iOS 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
You can use this method to fire a repeating timer without interrupting its regular firing schedule. If the timer is non-repeating, it is automatically invalidated after firing, even if its scheduled fire date has not arrived.

Availability
Available in iOS 2.0 and later.

See Also
- `invalidate` (page 1332)

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 the `isValid` (page 1333) method to verify that the timer is valid.

Availability
Available in iOS 2.0 and later.

See Also
- `setFireDate:` (page 1333)

Declared In
`NSTimer.h`

initWithFireDate:interval:target:selector:userInfo:repeats:

Initializes a new `NSTimer` object using the specified object and selector.

```
- (id)initWithFireDate:(NSDate *)date interval:(NSTimeInterval)seconds
    target:(id)target  selector:(SEL)aSelector  userInfo:(id)userInfo
    repeats:(BOOL)repeats
```

Parameters

date

The time at which the timer should first fire.

seconds

For a repeating timer, this parameter contains the number of seconds between firings of the timer. If *seconds* is less than or equal to `0.0`, this method chooses the nonnegative value of 0.1 milliseconds instead.

target

The object to which to send the message specified by *aSelector* when the timer fires. The target object is retained by the timer and released when the timer is invalidated.

aSelector

The message to send to *target* when the timer fires. The selector must have the following signature:

```
- (void)timerFireMethod:(NSTimer*)theTimer
```

The timer passes itself as the argument to this method.

userInfo

Custom user info for the timer. The object you specify is retained by the timer and released when the timer is invalidated. This parameter may be `nil`.

repeats

If `YES`, the timer will repeatedly reschedule itself until invalidated. If `NO`, the timer will be invalidated after it fires.

Return Value

The receiver, initialized such that, when added to a run loop, it will fire at *date* and then, if *repeats* is `YES`, every *seconds* after that.

Discussion

You must add the new timer to a run loop, using `addTimer:forMode:` (page 1107). Upon firing, the timer sends the message *aSelector* to *target*. (If the timer is configured to repeat, there is no need to subsequently re-add the timer to the run loop.)

Availability

Available in iOS 2.0 and later.

Declared In

`NSTimer.h`

invalidate

Stops the receiver from ever firing again and requests its removal from its run loop.

```
- (void)invalidate
```

Discussion

This method 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 1332) method returns or at some later point.

If it was configured with target and user info objects, the receiver releases its references to those objects as well.

Special Considerations

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.

Availability

Available in iOS 2.0 and later.

See Also

`- fire` (page 1331)

Related Sample Code

aurioTouch

GKRocket

GLSprite

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 still capable of firing or `NO` if the timer has been invalidated and is no longer capable of firing.

Availability

Available in iOS 2.0 and later.

Declared In

`NSTimer.h`

setFireDate:

Resets the firing time of the receiver to the specified date.

```
- (void)setFireDate:(NSDate *)date
```

Parameters

date

The new date at which to fire the receiver. If the new date is in the past, this method sets the fire time to the current time.

Discussion

You typically use this method to adjust the firing time of a repeating timer. Although resetting a timer's next firing time is a relatively expensive operation, it may be more efficient in some situations. For example, you could use it in situations where you want to repeat an action multiple times in the future, but at irregular time intervals. Adjusting the firing time of a single timer would likely incur less expense than creating multiple timer objects, scheduling each one on a run loop, and then destroying them.

You should not call this method on a timer that has been invalidated, which includes non-repeating timers that have already fired. You could potentially call this method on a non-repeating timer that had not yet fired, although you should always do so from the thread to which the timer is attached to avoid potential race conditions.

Availability
Available in iOS 2.0 and later.

See Also
- `fireDate` (page 1331)

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 iOS 2.0 and later.

Declared In
`NSTimer.h`

userInfo

Returns the receiver's *`userInfo`* object.

```
- (id)userInfo
```

Return Value
The receiver's *`userInfo`* object.

Discussion
Do not invoke this method after the timer is invalidated. Use `isValid` (page 1333) to test whether the timer is valid.

Availability
Available in iOS 2.0 and later.

See Also
+ `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` (page 1328)
+ `timerWithTimeInterval:target:selector:userInfo:repeats:` (page 1330)
- `invalidate` (page 1332)

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 iOS 2.0 and later.
Declared in	Foundation/NSTimeZone.h
Companion guide	Date and Time Programming Guide

Overview

`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 1342), `timeZoneWithName:data:` (page 1343), `timeZoneWithAbbreviation:` (page 1342), and `timeZoneForSecondsFromGMT:` (page 1341). The class also permits you to set the default time zone *within your application* (`setDefaultTimeZone:` (page 1340)). You can access this default time zone at any time with the `defaultTimeZone` (page 1338) class method, and with the `localTimeZone` (page 1339) 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.

Tasks

Creating and Initializing Time Zone Objects

`+ timeZoneWithAbbreviation:` (page 1342)
Returns the time zone object identified by a given abbreviation.

`+ timeZoneWithName:` (page 1342)
Returns the time zone object identified by a given ID.

`+ timeZoneWithName:data:` (page 1343)
Returns the time zone with a given ID whose data has been initialized using given data,

`+ timeZoneForSecondsFromGMT:` (page 1341)
Returns a time zone object offset from Greenwich Mean Time by a given number of seconds.

`- initWithName:` (page 1346)
Returns a time zone initialized with a given ID.

`- initWithName:data:` (page 1346)
Initializes a time zone with a given ID and time zone data.

`+ timeZoneDataVersion` (page 1341)
Returns the time zone data version.

Working with System Time Zones

`+ localTimeZone` (page 1339)
Returns an object that forwards all messages to the default time zone for the current application.

`+ defaultTimeZone` (page 1338)
Returns the default time zone for the current application.

`+ setDefaultTimeZone:` (page 1340)
Sets the default time zone for the current application to a given time zone.

`+ resetSystemTimeZone` (page 1339)
Resets the system time zone object cached by the application, if any.

`+ systemTimeZone` (page 1341)
Returns the time zone currently used by the system.

Getting Time Zone Information

`+ abbreviationDictionary` (page 1338)
Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

`+ knownTimeZoneNames` (page 1339)
Returns an array of strings listing the IDs of all the time zones known to the system.

+ `setAbbreviationDictionary:` (page 1340)
Sets the abbreviation dictionary to the specified dictionary.

Getting Information About a Specific Time Zone

- `abbreviation` (page 1344)
Returns the abbreviation for the receiver.
- `abbreviationForDate:` (page 1344)
Returns the abbreviation for the receiver at a given date.
- `name` (page 1349)
Returns the geopolitical region ID that identifies the receiver.
- `secondsFromGMT` (page 1350)
Returns the current difference in seconds between the receiver and Greenwich Mean Time.
- `secondsFromGMTForDate:` (page 1350)
Returns the difference in seconds between the receiver and Greenwich Mean Time at a given date.
- `data` (page 1344)
Returns the data that stores the information used by the receiver.

Comparing Time Zones

- `isEqualToTimeZone:` (page 1348)
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 1346)
Returns the description of the receiver.
- `localizedName:locale:` (page 1348)
Returns the name of the receiver localized for a given locale.

Getting Information About Daylight Saving

- `isDaylightSavingTime` (page 1347)
Returns a Boolean value that indicates whether the receiver is currently using daylight saving time.
- `daylightSavingTimeOffset` (page 1345)
Returns the current daylight saving time offset of the receiver.
- `isDaylightSavingTimeForDate:` (page 1347)
Returns a Boolean value that indicates whether the receiver uses daylight savings time at a given date.
- `daylightSavingTimeOffsetForDate:` (page 1345)
Returns the daylight saving time offset for a given date.

- `nextDaylightSavingTimeTransition` (page 1349)
  Returns the date of the next daylight saving time transition for the receiver.
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1349)
  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 iOS 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 1341) 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 1340) 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 1339).

Availability
Available in iOS 2.0 and later.

See Also
`+ localTimeZone` (page 1339)

+ setDefaultTimeZone: (page 1340)
+ systemTimeZone (page 1341)

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 iOS 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 1338)) and hold onto the returned object, it does not change if a subsequent invocation of setDefaultTimeZone: (page 1340) 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 iOS 2.0 and later.

See Also
+ defaultTimeZone (page 1338)
+ setDefaultTimeZone: (page 1340)

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 1341), `NSTimeZone` will attempt to redetermine the system time zone and a new object will be created and cached (see `systemTimeZone` (page 1341)).

Availability
Available in iOS 2.0 and later.

See Also
`+ systemTimeZone` (page 1341)

Declared In
`NSTimeZone.h`

setAbbreviationDictionary:

Sets the abbreviation dictionary to the specified dictionary.

```
+ (void)setAbbreviationDictionary:(NSDictionary *)dict
```

Parameters

dict

A dictionary containing key-value pairs for looking up time zone names given their abbreviations. The keys should be `NSString` objects containing the abbreviations; the values should be `NSString` objects containing their corresponding geopolitical region names.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
`+ defaultTimeZone` (page 1338)
`+ localTimeZone` (page 1339)

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 1339) to clear the cached object.

Availability
Available in iOS 2.0 and later.

See Also
`+ resetSystemTimeZone` (page 1339)

Declared In
NSTimeZone.h

timeZoneDataVersion

Returns the time zone data version.

```
+ (NSString *)timeZoneDataVersion
```

Return Value
A string containing the time zone data version.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

See Also
`+ timeZoneWithAbbreviation:` (page 1342)
`+ timeZoneWithName:` (page 1342)

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 iOS 2.0 and later.

See Also
`+ abbreviationDictionary` (page 1338)
`+ timeZoneForSecondsFromGMT:` (page 1341)
`+ timeZoneWithName:` (page 1342)

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 iOS 2.0 and later.

See Also

`+ timeZoneForSecondsFromGMT:` (page 1341)

`+ timeZoneWithAbbreviation:` (page 1342)

`+ knownTimeZoneNames` (page 1339)

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 1342) to get the time zone object for a given name.

Availability

Available in iOS 2.0 and later.

See Also

`+ timeZoneWithName:` (page 1342)

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 1344) with the current date as the argument.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
- isDaylightSavingTime (page 1347)
- isDaylightSavingTimeForDate: (page 1347)
- daylightSavingTimeOffsetForDate: (page 1345)

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 iOS 2.0 and later.

See Also
- isDaylightSavingTime (page 1347)
- daylightSavingTimeOffset (page 1345)
- isDaylightSavingTimeForDate: (page 1347)
- nextDaylightSavingTimeTransitionAfterDate: (page 1349)

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 iOS 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 1346) with the appropriate data object.

In Mac OS X v10.4 and earlier providing nil for the parameter would have caused a crash. In Mac OS X v10.5 and later, this now raises an invalid argument exception.

Availability
Available in iOS 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 1346) to get a time zone object.

In Mac OS X v10.4 and earlier providing nil for the parameter would have caused a crash. In Mac OS X v10.5 and later, this now raises an invalid argument exception.

Availability
Available in iOS 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 1347) with the current date as the argument.

Availability
Available in iOS 2.0 and later.

See Also
- `isDaylightSavingTimeForDate:` (page 1347)
- `daylightSavingTimeOffset` (page 1345)
- `daylightSavingTimeOffsetForDate:` (page 1345)
- `nextDaylightSavingTimeTransition` (page 1349)
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1349)

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 iOS 2.0 and later.

See Also
- `- isDaylightSavingTime` (page 1347)
- `- daylightSavingTimeOffset` (page 1345)
- `- daylightSavingTimeOffsetForDate:` (page 1345)
- `- nextDaylightSavingTimeTransitionAfterDate:` (page 1349)

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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

See Also
- `isDaylightSavingTime` (page 1347)
- `isDaylightSavingTimeForDate:` (page 1347)
- `nextDaylightSavingTimeTransitionAfterDate:` (page 1349)

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 iOS 2.0 and later.

See Also
- `isDaylightSavingTime` (page 1347)
- `isDaylightSavingTimeForDate:` (page 1347)

- nextDaylightSavingTimeTransition (page 1349)

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 iOS 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 iOS 2.0 and later.

Declared In
NSTimeZone.h

Constants

Time Zone Name Styles

Specify styles for presenting time zone names.

```
enum {
    NSTimeZoneNameStyleStandard,
    NSTimeZoneNameStyleShortStandard,
    NSTimeZoneNameStyleDaylightSaving,
    NSTimeZoneNameStyleShortDaylightSaving
};
typedef NSInteger NSTimeZoneNameStyle;
```

Constants

`NSTimeZoneNameStyleStandard`
Specifies a standard name style. For example, “Central Standard Time” for Central Time.
Available in iOS 2.0 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleShortStandard`
Specifies a short name style. For example, “CST” for Central Time.
Available in iOS 2.0 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleDaylightSaving`
Specifies a daylight saving name style. For example, “Central Daylight Time” for Central Time.
Available in iOS 2.0 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleShortDaylightSaving`
Specifies a short daylight saving name style. For example, “CDT” for Central Time.
Available in iOS 2.0 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleGeneric`
Specifies a generic name style. For example, “Central Time” for Central Time.
Available in iOS 4.0 and later.
Declared in `NSTimeZone.h`.

`NSTimeZoneNameStyleShortGeneric`
Specifies a generic time zone name. For example, “CT” for Central Time.
Available in iOS 4.0 and later.
Declared in `NSTimeZone.h`.

Declared In
`NSTimeZone.h`

Notifications

NSSystemTimeZoneDidChangeNotification

Sent when the time zone changed.

Availability
Available in iOS 2.0 and later.

Declared In
`NSTimeZone.h`

NSUndoManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	Foundation/NSUndoManager.h
Companion guide	Undo Architecture

Overview

`NSUndoManager` is a general-purpose recorder of operations for undo and redo.

You register an undo operation by specifying the object that's changing (or the owner of that object), along with a method to invoke to revert its state, and the arguments for that method. When performing undo an `NSUndoManager` saves the operations reverted so that you can redo the undos.

`NSUndoManager` is implemented as a class of the Foundation framework because executables other than applications might want to revert changes to their states. For example, you might have an interactive command-line tool with undo and redo commands, or there could be distributed object implementations that can revert operations "over the wire." However, users typically see undo and redo as application features. UIKit implements undo and redo in its text view object and makes it easy to implement it in objects along the responder chain (see `UIResponder`).

Tasks

Registering Undo Operations

- `registerUndoWithTarget:selector:object:` (page 1363)
  Records a single undo operation for a given target, so that when an undo is performed it is sent a specified selector with a given object as the sole argument.
- `prepareWithInvocationTarget:` (page 1361)
  Prepares the receiver for invocation-based undo with the given target as the subject of the next undo operation and returns `self`.

Checking Undo Ability

- `canUndo` (page 1357)
  Returns a Boolean value that indicates whether the receiver has any actions to undo.
- `canRedo` (page 1356)
  Returns a Boolean value that indicates whether the receiver has any actions to redo.

Performing Undo and Redo

- `undo` (page 1367)
  Closes the top-level undo group if necessary and invokes `undoNestedGroup` (page 1369).
- `undoNestedGroup` (page 1369)
  Performs the undo operations in the last undo group (whether top-level or nested), recording the operations on the redo stack as a single group.
- `redo` (page 1361)
  Performs the operations in the last group on the redo stack, if there are any, recording them on the undo stack as a single group.

Limiting the Undo Stack

- `setLevelsOfUndo:` (page 1366)
  Sets the maximum number of top-level undo groups the receiver holds.
- `levelsOfUndo` (page 1361)
  Returns the maximum number of top-level undo groups the receiver holds.

Creating Undo Groups

- `beginUndoGrouping` (page 1356)
  Marks the beginning of an undo group.
- `endUndoGrouping` (page 1358)
  Marks the end of an undo group.
- `groupsByEvent` (page 1359)
  Returns a Boolean value that indicates whether the receiver automatically creates undo groups around each pass of the run loop.
- `setGroupsByEvent:` (page 1366)
  Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.
- `groupingLevel` (page 1359)
  Returns the number of nested undo groups (or redo groups, if Redo was invoked last) in the current event loop.

Enabling and Disabling Undo

- `disableUndoRegistration` (page 1357)
  Disables the recording of undo operations, whether by `registerUndoWithTarget:selector:object:` (page 1363) or by invocation-based undo.
- `enableUndoRegistration` (page 1358)
  Enables the recording of undo operations.
- `isUndoRegistrationEnabled` (page 1360)
  Returns a Boolean value that indicates whether the recording of undo operations is enabled.

Checking Whether Undo or Redo Is Being Performed

- `isUndoing` (page 1360)
  Returns a Boolean value that indicates whether the receiver is in the process of performing its `undo` (page 1367) or `undoNestedGroup` (page 1369) method.
- `isRedoing` (page 1359)
  Returns a Boolean value that indicates whether the receiver is in the process of performing its `redo` (page 1361) method.

Clearing Undo Operations

- `removeAllActions` (page 1364)
  Clears the undo and redo stacks and re-enables the receiver.
- `removeAllActionsWithTarget:` (page 1364)
  Clears the undo and redo stacks of all operations involving the specified target as the recipient of the undo message.

Managing the Action Name

- `setActionName:` (page 1365)
  Sets the name of the action associated with the Undo or Redo command.
- `redoActionName` (page 1362)
  Returns the name identifying the redo action.
- `undoActionName` (page 1367)
  Returns the name identifying the undo action.

Getting and Localizing the Menu Item Title

- `redoMenuItemTitle` (page 1362)
  Returns the complete title of the Redo menu command, for example, “Redo Paste.”
- `undoMenuItemTitle` (page 1368)
  Returns the complete title of the Undo menu command, for example, “Undo Paste.”

- `redoMenuTitleForUndoActionName:` (page 1363)
  Returns the complete, localized title of the Redo menu command for the action identified by the given name.
- `undoMenuTitleForUndoActionName:` (page 1368)
  Returns the complete, localized title of the Undo menu command for the action identified by the given name.

Working with Run Loops

- `runLoopModes` (page 1365)
  Returns the modes governing the types of input handled during a cycle of the run loop.
- `setRunLoopModes:` (page 1367)
  Sets the modes that determine the types of input handled during a cycle of the run loop.

Instance Methods

beginUndoGrouping

Marks the beginning of an undo group.

```
- (void)beginUndoGrouping
```

Discussion
All individual undo operations before a subsequent `endUndoGrouping` (page 1358) message are grouped together and reversed by a later `undo` (page 1367) message. By default undo groups are begun automatically at the start of the event loop, but you can begin your own undo groups with this method, and nest them within other groups.

This method posts an `NSUndoManagerCheckpointNotification` (page 1370) unless a top-level undo is in progress. It posts an `NSUndoManagerDidOpenUndoGroupNotification` (page 1370) if a new group was successfully created.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

canRedo

Returns a Boolean value that indicates whether the receiver has any actions to redo.

```
- (BOOL)canRedo
```

Return Value
`YES` if the receiver has any actions to redo, otherwise `NO`.

Discussion
Because any undo operation registered clears the redo stack, this method posts an `NSUndoManagerCheckpointNotification` (page 1370) to allow clients to apply their pending operations before testing the redo stack.

Availability
Available in iOS 3.0 and later.

See Also
- `canUndo` (page 1357)
- `redo` (page 1361)

Declared In
`NSUndoManager.h`

canUndo

Returns a Boolean value that indicates whether the receiver has any actions to undo.

```
- (BOOL)canUndo
```

Return Value
`YES` if the receiver has any actions to undo, otherwise `NO`.

Discussion
The return value does not mean you can safely invoke `undo` (page 1367) or `undoNestedGroup` (page 1369)—you may have to close open undo groups first.

Availability
Available in iOS 3.0 and later.

See Also
- `canRedo` (page 1356)
- `enableUndoRegistration` (page 1358)
- `registerUndoWithTarget:selector:object:` (page 1363)

Declared In
`NSUndoManager.h`

disableUndoRegistration

Disables the recording of undo operations, whether by `registerUndoWithTarget:selector:object:` (page 1363) or by invocation-based undo.

```
- (void)disableUndoRegistration
```

Discussion
This method can be invoked multiple times by multiple clients. The `enableUndoRegistration` (page 1358) method must be invoked an equal number of times to re-enable undo registration.

Availability
Available in iOS 3.0 and later.

See Also
- `enableUndoRegistration` (page 1358)
- `isUndoRegistrationEnabled` (page 1360)

Declared In
`NSUndoManager.h`

enableUndoRegistration

Enables the recording of undo operations.

```
- (void)enableUndoRegistration
```

Discussion
Because undo registration is enabled by default, it is often used to balance a prior `disableUndoRegistration` (page 1357) message. Undo registration isn't actually re-enabled until an enable message balances the last disable message in effect. Raises an `NSInternalInconsistencyException` if invoked while no `disableUndoRegistration` (page 1357) message is in effect.

Availability
Available in iOS 3.0 and later.

See Also
- `disableUndoRegistration` (page 1357)
- `isUndoRegistrationEnabled` (page 1360)

Declared In
`NSUndoManager.h`

endUndoGrouping

Marks the end of an undo group.

```
- (void)endUndoGrouping
```

Discussion
All individual undo operations back to the matching `beginUndoGrouping` (page 1356) message are grouped together and reversed by a later `undo` (page 1367) or `undoNestedGroup` (page 1369) message. Undo groups can be nested, thus providing functionality similar to nested transactions. Raises an `NSInternalInconsistencyException` if there's no `beginUndoGrouping` (page 1356) message in effect.

This method posts an `NSUndoManagerCheckpointNotification` (page 1370) and an `NSUndoManagerWillCloseUndoGroupNotification` (page 1371) just before the group is closed.

Availability
Available in iOS 3.0 and later.

See Also
- `levelsOfUndo` (page 1361)

Declared In
`NSUndoManager.h`

groupingLevel

Returns the number of nested undo groups (or redo groups, if Redo was invoked last) in the current event loop.

```
- (NSInteger)groupingLevel
```

Return Value
An integer indicating the number of nested groups. If 0 is returned, there is no open undo or redo group.

Availability
Available in iOS 3.0 and later.

See Also
- `levelsOfUndo` (page 1361)
- `setLevelsOfUndo:` (page 1366)

Declared In
`NSUndoManager.h`

groupsByEvent

Returns a Boolean value that indicates whether the receiver automatically creates undo groups around each pass of the run loop.

```
- (BOOL)groupsByEvent
```

Return Value
`YES` if the receiver automatically creates undo groups around each pass of the run loop, otherwise `NO`.

Discussion
The default is `YES`.

Availability
Available in iOS 3.0 and later.

See Also
- `beginUndoGrouping` (page 1356)
- `setGroupsByEvent:` (page 1366)

Declared In
`NSUndoManager.h`

isRedoing

Returns a Boolean value that indicates whether the receiver is in the process of performing its `redo` (page 1361) method.

```
- (BOOL)isRedoing
```

Return Value
`YES` if the method is being performed, otherwise `NO`.

Availability
Available in iOS 3.0 and later.

See Also
- `isUndoing` (page 1360)

Declared In
`NSUndoManager.h`

isUndoing

Returns a Boolean value that indicates whether the receiver is in the process of performing its `undo` (page 1367) or `undoNestedGroup` (page 1369) method.

```
- (BOOL)isUndoing
```

Return Value
`YES` if the method is being performed, otherwise `NO`.

Availability
Available in iOS 3.0 and later.

See Also
- `isRedoing` (page 1359)

Declared In
`NSUndoManager.h`

isUndoRegistrationEnabled

Returns a Boolean value that indicates whether the recording of undo operations is enabled.

```
- (BOOL)isUndoRegistrationEnabled
```

Return Value
`YES` if registration is enabled; otherwise, `NO`.

Discussion
Undo registration is enabled by default.

Availability
Available in iOS 3.0 and later.

See Also
- `disableUndoRegistration` (page 1357)
- `enableUndoRegistration` (page 1358)

Declared In
`NSUndoManager.h`

levelsOfUndo

Returns the maximum number of top-level undo groups the receiver holds.

```
- (NSUInteger)levelsOfUndo
```

Return Value
An integer specifying the number of undo groups. A limit of `0` indicates no limit, so old undo groups are never dropped.

Discussion
When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. The default is `0`.

Availability
Available in iOS 3.0 and later.

See Also
- `enableUndoRegistration` (page 1358)
- `setLevelsOfUndo:` (page 1366)

Declared In
`NSUndoManager.h`

prepareWithInvocationTarget:

Prepares the receiver for invocation-based undo with the given target as the subject of the next undo operation and returns `self`.

```
- (id)prepareWithInvocationTarget:(id)target
```

Parameters
target
The target of the undo operation.

Return Value
`self`.

Discussion
See Registering Undo Operations for more information.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

redo

Performs the operations in the last group on the redo stack, if there are any, recording them on the undo stack as a single group.

```
- (void)redo
```

Discussion
Raises an `NSInternalInconsistencyException` if the method is invoked during an undo operation.

This method posts an `NSUndoManagerCheckpointNotification` (page 1370) and `NSUndoManagerWillRedoChangeNotification` (page 1371) before it performs the redo operation, and it posts the `NSUndoManagerDidRedoChangeNotification` (page 1370) after it performs the redo operation.

Availability
Available in iOS 3.0 and later.

See Also
- `registerUndoWithTarget:selector:object:` (page 1363)

Declared In
`NSUndoManager.h`

redoActionName

Returns the name identifying the redo action.

```
- (NSString *)redoActionName
```

Return Value
The redo action name. Returns an empty string (`@""`) if no action name has been assigned or if there is nothing to redo.

Discussion
For example, if the menu title is “Redo Delete,” the string returned is “Delete.”

Availability
Available in iOS 3.0 and later.

See Also
- `setActionName:` (page 1365)
- `undoActionName` (page 1367)

Declared In
`NSUndoManager.h`

redoMenuItemTitle

Returns the complete title of the Redo menu command, for example, “Redo Paste.”

```
- (NSString *)redoMenuItemTitle
```

Return Value
The menu item title.

Discussion
Returns “Redo” if no action name has been assigned or `nil` if there is nothing to redo.

Availability
Available in iOS 3.0 and later.

See Also
- `undoMenuItemTitle` (page 1368)

Declared In
`NSUndoManager.h`

redoMenuTitleForUndoActionName:

Returns the complete, localized title of the Redo menu command for the action identified by the given name.

```
- (NSString *)redoMenuTitleForUndoActionName:(NSString *)actionName
```

Parameters

actionName
The name of the undo action.

Return Value
The localized title of the redo menu item.

Discussion
Override this method if you want to customize the localization behavior. This method is invoked by `redoMenuItemTitle` (page 1362).

Availability
Available in iOS 3.0 and later.

See Also
- `undoMenuTitleForUndoActionName:` (page 1368)

Declared In
`NSUndoManager.h`

registerUndoWithTarget:selector:object:

Records a single undo operation for a given target, so that when an undo is performed it is sent a specified selector with a given object as the sole argument.

```
- (void)registerUndoWithTarget:(id)target selector:(SEL)aSelector object:(id)anObject
```

Parameters

target
The target of the undo operation.

aSelector
The selector for the undo operation.

anObject
The argument sent with the selector.

Discussion
Also clears the redo stack. Does not retain *target*, but does retain *anObject*. See Registering Undo Operations for more information.

Raises an `NSInternalInconsistencyException` if invoked when no undo group has been established using `beginUndoGrouping` (page 1356). Undo groups are normally set by default, so you should rarely need to begin a top-level undo group explicitly.

Availability
Available in iOS 3.0 and later.

See Also
- `undoNestedGroup` (page 1369)
- `groupingLevel` (page 1359)

Declared In
`NSUndoManager.h`

removeAllActions

Clears the undo and redo stacks and re-enables the receiver.

```
- (void)removeAllActions
```

Availability
Available in iOS 3.0 and later.

See Also
- `enableUndoRegistration` (page 1358)
- `removeAllActionsWithTarget:` (page 1364)

Declared In
`NSUndoManager.h`

removeAllActionsWithTarget:

Clears the undo and redo stacks of all operations involving the specified target as the recipient of the undo message.

```
- (void)removeAllActionsWithTarget:(id)target
```

Parameters
target
The recipient of the undo messages to be removed.

Discussion
Doesn't re-enable the receiver if it's disabled. An object that shares an `NSUndoManager` with other clients should invoke this message in its implementation of `dealloc`.

Availability
Available in iOS 3.0 and later.

See Also
- `enableUndoRegistration` (page 1358)
- `removeAllActions` (page 1364)

Declared In
NSUndoManager.h

runLoopModes

Returns the modes governing the types of input handled during a cycle of the run loop.

```
- (NSArray *)runLoopModes
```

Return Value
An array of string constants specifying the current run-loop modes.

Discussion
By default, the sole run-loop mode is `NSDefaultRunLoopMode` (which excludes data from `NSConnection` objects).

Availability
Available in iOS 3.0 and later.

See Also
- `setRunLoopModes:` (page 1367)
- `performSelector:target:argument:order:modes:` (page 1110) (`NSRunLoop`)

Declared In
NSUndoManager.h

setActionName:

Sets the name of the action associated with the Undo or Redo command.

```
- (void)setActionName:(NSString *)actionName
```

Parameters
actionName
The name of the action.

Discussion
If *actionName* is an empty string, the action name currently associated with the menu command is removed. There is no effect if *actionName* is `nil`.

Availability
Available in iOS 3.0 and later.

See Also
- `redoActionName` (page 1362)
- `undoActionName` (page 1367)

Declared In
NSUndoManager.h

setGroupsByEvent:

Sets a Boolean value that specifies whether the receiver automatically groups undo operations during the run loop.

```
- (void)setGroupsByEvent:(BOOL)flag
```

Parameters

flag

If YES, the receiver creates undo groups around each pass through the run loop; if NO it doesn't.

Discussion

The default is YES. If you turn automatic grouping off, you must close groups explicitly before invoking either undo (page 1367) or undoNestedGroup (page 1369).

Availability

Available in iOS 3.0 and later.

See Also

- groupingLevel (page 1359)
- groupsByEvent (page 1359)

Declared In

NSUndoManager.h

setLevelsOfUndo:

Sets the maximum number of top-level undo groups the receiver holds.

```
- (void)setLevelsOfUndo:(NSUInteger)anInt
```

Parameters

anInt

The maximum number of undo groups. A limit of 0 indicates no limit, so that old undo groups are never dropped.

Discussion

When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. The default is 0.

If invoked with a limit below the prior limit, old undo groups are immediately dropped.

Availability

Available in iOS 3.0 and later.

See Also

- enableUndoRegistration (page 1358)
- levelsOfUndo (page 1361)

Declared In

NSUndoManager.h

setRunLoopModes:

Sets the modes that determine the types of input handled during a cycle of the run loop.

```
- (void)setRunLoopModes:(NSArray *)modes
```

Parameters

modes

An array of string constants specifying the run-loop modes to set.

Discussion

By default, the sole run-loop mode is `NSDefaultRunLoopMode` (which excludes data from `NSConnection` objects). With this method, you could limit the input to data received during a mouse-tracking session by setting the mode to `NSEventTrackingRunLoopMode`, or you could limit it to data received from a modal panel with `NSModalPanelRunLoopMode`.

Availability

Available in iOS 3.0 and later.

See Also

- `runLoopModes` (page 1365)
- `performSelector:target:argument:order:modes:` (page 1110) (`NSRunLoop`)

Declared In

`NSUndoManager.h`

undo

Closes the top-level undo group if necessary and invokes `undoNestedGroup` (page 1369).

```
- (void)undo
```

Discussion

This method also invokes `endUndoGrouping` (page 1358) if the nesting level is 1. Raises an `NSInternalInconsistencyException` if more than one undo group is open (that is, if the last group isn't at the top level).

This method posts an `NSUndoManagerCheckpointNotification` (page 1370).

Availability

Available in iOS 3.0 and later.

See Also

- `enableUndoRegistration` (page 1358)
- `groupingLevel` (page 1359)

Declared In

`NSUndoManager.h`

undoActionName

Returns the name identifying the undo action.

```
- (NSString *)undoActionName
```

Return Value
The undo action name. Returns an empty string (`@""`) if no action name has been assigned or if there is nothing to undo.

Discussion
For example, if the menu title is “Undo Delete,” the string returned is “Delete.”

Availability
Available in iOS 3.0 and later.

See Also
- `redoActionName` (page 1362)
- `setActionName:` (page 1365)

Declared In
`NSUndoManager.h`

undoMenuItemTitle

Returns the complete title of the Undo menu command, for example, “Undo Paste.”

```
- (NSString *)undoMenuItemTitle
```

Return Value
The menu item title.

Discussion
Returns “Undo” if no action name has been assigned or `nil` if there is nothing to undo.

Availability
Available in iOS 3.0 and later.

See Also
- `redoMenuItemTitle` (page 1362)

Declared In
`NSUndoManager.h`

undoMenuTitleForUndoActionName:

Returns the complete, localized title of the Undo menu command for the action identified by the given name.

```
- (NSString *)undoMenuTitleForUndoActionName:(NSString *)actionName
```

Parameters
actionName
The name of the undo action.

Return Value
The localized title of the undo menu item.

Discussion
Override this method if you want to customize the localization behavior. This method is invoked by `undoMenuItemTitle` (page 1368).

Availability
Available in iOS 3.0 and later.

See Also
`- redoMenuTitleForUndoActionName:` (page 1363)

Declared In
`NSUndoManager.h`

undoNestedGroup

Performs the undo operations in the last undo group (whether top-level or nested), recording the operations on the redo stack as a single group.

```
- (void)undoNestedGroup
```

Discussion
Raises an `NSInternalInconsistencyException` if any undo operations have been registered since the last `enableUndoRegistration` (page 1358) message.

This method posts an `NSUndoManagerCheckpointNotification` (page 1370) and `NSUndoManagerWillUndoChangeNotification` (page 1371) before it performs the undo operation, and it posts an `NSUndoManagerDidUndoChangeNotification` (page 1371) after it performs the undo operation.

Availability
Available in iOS 3.0 and later.

See Also
`- undo` (page 1367)

Declared In
`NSUndoManager.h`

Constants

NSUndoCloseGroupingRunLoopOrdering

`NSUndoManager` provides this constant as a convenience; you can use it to compare to values returned by some `NSUndoManager` methods.

```
enum {
   NSUndoCloseGroupingRunLoopOrdering = 350000
};
```

Constants

`NSUndoCloseGroupingRunLoopOrdering`

Used with `NSRunLoop`'s `performSelector:target:argument:order:modes:` (page 1110).

Available in iOS 3.0 and later.

Declared in `NSUndoManager.h`.

Declared In

`NSUndoManager.h`

Notifications

NSUndoManagerCheckpointNotification

Posted whenever an `NSUndoManager` object opens or closes an undo group (except when it opens a top-level group) and when checking the redo stack in `canRedo` (page 1356). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerDidOpenUndoGroupNotification

Posted whenever an `NSUndoManager` object opens an undo group, which occurs in the implementation of the `beginUndoGrouping` (page 1356) method. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerDidRedoChangeNotification

Posted just after an `NSUndoManager` object performs a redo operation (`redo` (page 1361)). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerDidUndoChangeNotification

Posted just after an `NSUndoManager` object performs an undo operation. If you invoke `undo` (page 1367) or `undoNestedGroup` (page 1369), this notification is posted. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillCloseUndoGroupNotification

Posted before an `NSUndoManager` object closes an undo group, which occurs in the implementation of the `endUndoGrouping` (page 1358) method. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillRedoChangeNotification

Posted just before an `NSUndoManager` object performs a redo operation (`redo` (page 1361)). The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSUndoManagerWillUndoChangeNotification

Posted just before an `NSUndoManager` object performs an undo operation. If you invoke `undo` (page 1367) or `undoNestedGroup` (page 1369), this notification is posted. The notification object is the `NSUndoManager` object. This notification does not contain a *`userInfo`* dictionary.

Availability
Available in iOS 3.0 and later.

Declared In
`NSUndoManager.h`

NSURL Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURL.h
Companion guide	URL Loading System Programming Guide
Related sample code	AddMusic BonjourWeb GKTank MoviePlayer

Overview

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 1393) 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:` (page 1200) and NSData provides `dataWithContentsOfURL:` (page 262).

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 return `nil`, which you must be prepared to handle. If you are creating NSURL objects using file system paths, you should use `fileURLWithPath:` (page 1378) or `initFileURLWithPath:` (page 1387), 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 classes `NSURLConnection` and `NSURLDownload` define methods useful for loading URL resources in the background. See *URL Loading System Programming Guide* for more information

See also NSURL Additions Reference in the Application Kit framework, which add methods supporting pasteboards.

NSURL is "toll-free bridged" with its Core Foundation counterpart, *CFURL Reference*. 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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

NSURLHandleClient
- `URLHandleResourceDidBeginLoading:`
- `URLHandleResourceDidCancelLoading:`
- `URLHandleResourceDidFinishLoading:`
- `URLHandle:resourceDataDidBecomeAvailable:`
- `URLHandle:resourceDidFailLoadingWithReason:`

Tasks

Creating an NSURL

`- initWithScheme:host:path:` (page 1388)
Initializes a newly created NSURL with a specified scheme, host, and path.

`+ URLWithString:` (page 1381)
Creates and returns an NSURL object initialized with a provided string.

`- initWithString:` (page 1388)
Initializes an NSURL object with a provided string.

`+ URLWithString:relativeToURL:` (page 1381)
Creates and returns an NSURL object initialized with a base URL and a relative string.

`- initWithString:relativeToURL:` (page 1389)
Initializes an NSURL object with a base URL and a relative string.

+ fileURLWithPath:isDirectory: (page 1379)
Initializes and returns a newly created NSURL object as a file URL with a specified path.

- initFileURLWithPath:isDirectory: (page 1387)
Initializes a newly created NSURL referencing the local file or directory at *path*.

+ fileURLWithPath: (page 1378)
Initializes and returns a newly created NSURL object as a file URL with a specified path.

- initFileURLWithPath: (page 1387)
Initializes a newly created NSURL referencing the local file or directory at *path*.

+ fileURLWithPathComponents: (page 1379)
Initializes and returns a newly created NSURL object as a file URL with specified path components.

+ URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error: (page 1380)
Returns a new URL made by resolving bookmark data.

- initByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error: (page 1386)
Initializes a newly created NSURL that points to a location specified by resolving bookmark data.

Identifying and Comparing Objects

- isEqual: (page 1389)
Returns a Boolean value that indicates whether the receiver and a given object are equal.

Querying an NSURL

- checkResourceIsReachableAndReturnError: (page 1384)
Returns whether the resource pointed to by a file URL can be reached.

- isFileReferenceURL (page 1390)
Returns whether the URL is a file reference URL.

- isFileURL (page 1390)
Returns whether the receiver uses the file scheme.

Accessing the Parts of the URL

- absoluteString (page 1382)
Returns the string for the receiver as if it were an absolute URL.

- absoluteURL (page 1383)
Returns an absolute URL that refers to the same resource as the receiver.

- baseURL (page 1383)
Returns the base URL of the receiver.

- fragment (page 1385)
Returns the fragment of a URL conforming to RFC 1808.

- host (page 1386)
Returns the host of a URL conforming to RFC 1808.

- `lastPathComponent` (page 1390)
  Returns the last path component of a file URL.
- `parameterString` (page 1391)
  Returns the parameter string of a URL conforming to RFC 1808.
- `password` (page 1391)
  Returns the password of a URL conforming to RFC 1808.
- `path` (page 1391)
  Returns the path of a URL conforming to RFC 1808.
- `pathComponents` (page 1392)
  Returns the individual path components of a file URL in an array.
- `pathExtension` (page 1392)
  Returns the path extension of a file URL.
- `port` (page 1392)
  Returns the port number of a URL conforming to RFC 1808.
- `query` (page 1392)
  Returns the query of a URL conforming to RFC 1808.
- `relativePath` (page 1393)
  Returns the path of a URL conforming to RFC 1808, without resolving against the receiver's base URL.
- `relativeString` (page 1393)
  Returns a string representation of the relative portion of the URL.
- `resourceSpecifier` (page 1393)
  Returns the resource specifier of the URL.
- `scheme` (page 1394)
  Returns the scheme of the URL.
- `standardizedURL` (page 1396)
  Returns a new NSURL object with any instances of " . . " or " . " removed from its path.
- `user` (page 1398)
  Returns the user portion of a URL conforming to RFC 1808.

Modifying and Converting a File URL

- `filePathURL` (page 1384)
  Returns a new file path URL that points to the same resource as the original URL.
- `fileReferenceURL` (page 1384)
  Returns a new file reference URL that points to the same resource as the original URL.
- `URLByAppendingPathComponent:` (page 1396)
  Returns a new URL made by appending a path component to the original URL.
- `URLByAppendingPathExtension:` (page 1396)
  Returns a new URL made by appending a path extension to the original URL.
- `URLByDeletingLastPathComponent` (page 1397)
  Returns a new URL made by deleting the last path component from the original URL.
- `URLByDeletingPathExtension` (page 1397)
  Returns a new URL made by deleting the path extension, if any, from the original URL.

- URLByResolvingSymlinksInPath (page 1398)
  Returns a new URL that points to the same resource as the original URL and includes no symbolic links.
- URLByStandardizingPath (page 1398)
  Returns a new URL that points to the same resource as the original URL and is an absolute path.

Working with Bookmark Data

+ bookmarkDataWithContentsOfURL:error: (page 1377)
  Initializes and returns bookmark data derived from an alias file pointed to by a specified URL.
- bookmarkDataWithOptions:includingResourceValuesForKeys:relativeToURL:error: (page 1383)
  Returns bookmark data for the URL, created with specified options and resource values.
+ writeBookmarkData:toURL:options:error: (page 1382)
  Creates an alias file on disk at a specified location with specified bookmark data.

Getting and Setting File System Resource Properties

- getResourceValue:forKey:error: (page 1385)
  Returns the resource value for the property identified by a given key.
- resourceValuesForKeys:error: (page 1394)
  Returns the resource values for the properties identified by specified array of keys.
- setResourceValue:forKey:error: (page 1394)
  Sets the resource property of the URL specified by a given key to a given value.
+ resourceValuesForKeys:fromBookmarkData: (page 1380)
  Returns the resource values for properties identified by a specified array of keys contained in specified bookmark data.
- setResourceValues:error: (page 1395)
  Sets resource properties of the URL specified by a given set of keys to a given set of values.

Class Methods

bookmarkDataWithContentsOfURL:error:

Initializes and returns bookmark data derived from an alias file pointed to by a specified URL.

```
+ (NSData *)bookmarkDataWithContentsOfURL:(NSURL *)bookmarkFileURL error:(NSError **)error
```

Parameters

bookmarkFileURL
  The URL that points to the alias file.

error
  The error that occurred in the case that the bookmark data cannot be derived.

Return Value
The bookmark data for the alias file.

Discussion
If *bookmarkFileURL* points to an alias file created prior to Mac OS X v10.6 that contains Alias Manager information but no bookmark data, this method synthesizes bookmark data for the file.

This method returns `nil` if bookmark data cannot be created.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

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 1267).

Passing `nil` for this parameter produces an exception.

Return Value
An NSURL object initialized with *path*.

Discussion
This method assumes that *path* is a directory if it ends with a slash. If *path* does not end with a slash, the method examines the file system to determine if *path* is a file or a directory. If *path* exists in the file system and is a directory, the method appends a trailing slash. If *path* does not exist in the file system, the method assumes that it represents a file and does not append a trailing slash.

As an alternative, consider using `fileURLWithPath:isDirectory:` (page 1379), which allows you to explicitly specify whether the returned `NSURL` object represents a file or directory.

Availability
Available in iOS 2.0 and later.

See Also
`initFileURLWithPath:` (page 1387)

Related Sample Code
GKTank
MoviePlayer

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 1267).

Passing `nil` for this parameter produces an exception.

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 iOS 2.0 and later.

See Also
`initFileURLWithPath:` (page 1387)

Declared In
`NSURL.h`

fileURLWithPathComponents:

Initializes and returns a newly created NSURL object as a file URL with specified path components.

```
+ (NSURL *)fileURLWithPathComponents:(NSArray *)components
```

Parameters

components

An array of path components.

Passing `nil` for this parameter produces an exception.

Return Value
An NSURL object initialized with *components*.

Discussion
The path components are separated by a forward slash in the returned URL.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

resourceValuesForKeys:fromBookmarkData:

Returns the resource values for properties identified by a specified array of keys contained in specified bookmark data.

```
+ (NSDictionary *)resourceValuesForKeys:(NSArray *)keys fromBookmarkData:(NSData
    *)bookmarkData
```

Parameters

keys

An array of names of URL resource properties.

bookmarkData

The bookmark data the resource values are derived from.

Return Value

A dictionary of the requested resource values contained in *bookmarkData*.

Availability

Available in iOS 4.0 and later.

See Also

"Common File System Resource Keys" (page 1399)

"File Property Keys" (page 1402)

"Volume Property Keys" (page 1403)

Declared In

NSURL.h

URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:

Returns a new URL made by resolving bookmark data.

```
+ (id)URLByResolvingBookmarkData:(NSData *)bookmarkData
    options:(NSURLBookmarkResolutionOptions)options relativeToURL:(NSURL
    *)relativeURL bookmarkDataIsStale:(BOOL *)isStale error:(NSError **)error
```

Parameters

bookmarkData

The bookmark data the URL is derived from.

options

Options taken into account when resolving the bookmark data.

relativeURL

The base URL that the bookmark data is relative to.

isStale

If YES, the bookmark data is stale.

error

The error that occurred in the case that the URL cannot be created.

Return Value

A new URL made by resolving *bookmarkData*.

Availability

Available in iOS 4.0 and later.

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 iOS 2.0 and later.

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

Declared In
NSURL.h

writeBookmarkData:toURL:options:error:

Creates an alias file on disk at a specified location with specified bookmark data.

```
+ (BOOL)writeBookmarkData:(NSData *)bookmarkData toURL:(NSURL *)bookmarkFileURL
    options:(NSURLBookmarkFileCreationOptions)options error:(NSError **)error
```

Parameters

bookmarkData

The bookmark data containing information for the alias file.

bookmarkFileURL

The desired location of the alias file.

options

Options taken into account when creating the alias file.

error

The error that occurred in the case that the alias file cannot be created.

Return Value
`YES` if the alias file is successfully created; otherwise, `NO`.

Discussion
This method will produce an error if *bookmarkData* was not created with the `NSURLBookmarkCreationSuitableForBookmarkFile` option.

If *bookmarkFileURL* points to a directory, the alias file will be created in that directory with its name derived from the information in *bookmarkData*. If *bookmarkFileURL* points to a file, the alias file will be created with the location and name indicated by *bookmarkFileURL*, and its extension will be changed to `.alias` if it is not already.

Availability
Available in iOS 4.0 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
`NSURL.h`

bookmarkDataWithOptions:includingResourceValuesForKeys:relativeToURL:error:

Returns bookmark data for the URL, created with specified options and resource values.

```
- (NSData *)bookmarkDataWithOptions:(NSURLBookmarkCreationOptions)options
    includingResourceValuesForKeys:(NSArray *)keys relativeToURL:(NSURL *)relativeURL
     error:(NSError **)error
```

Parameters

options
Options taken into account when creating the bookmark data.

keys
An array of names of URL resource properties.

relativeURL
The URL that the bookmark data will be relative to.

error
The error that occurred in the case that the bookmark data cannot be created.

Return Value
The bookmark data for the URL.

Availability
Available in iOS 4.0 and later.

Declared In
NSURL.h

checkResourceIsReachableAndReturnError:

Returns whether the resource pointed to by a file URL can be reached.

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

Parameters

error
The error that occurred in the case that the resource cannot be reached.

Return Value
YES if the resource is reachable; otherwise, NO.

Availability
Available in iOS 4.0 and later.

Declared In
NSURL.h

filePathURL

Returns a new file path URL that points to the same resource as the original URL.

```
- (NSURL *)filePathURL
```

Return Value
The new file path URL.

Discussion
If the original URL is a file reference URL, this method converts it to a file path URL. If the original URL is a file path URL, the returned URL is identical. If the original URL is not a file URL, this method returns nil.

Availability
Available in iOS 4.0 and later.

Declared In
NSURL.h

fileReferenceURL

Returns a new file reference URL that points to the same resource as the original URL.

```
- (NSURL *)fileReferenceURL
```

Return Value
The new file reference URL.

Discussion
If the original URL is a file path URL, this method converts it to a file reference URL. If the original URL is a file reference URL, the returned URL is identical. If the original URL is not a file URL, this method returns `nil`.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Declared In
`NSURL.h`

getResourceValue:forKey:error:

Returns the resource value for the property identified by a given key.

```
- (BOOL)getResourceValue:(id *)value forKey:(NSString *)key error:(NSError **)error
```

Parameters

value
The value for the property identified by *key*.

key
The name of one of the URL's resource properties.

error
The error that occurred in the case that the resource value cannot be retrieved.

Return Value
`YES` if *value* is successfully populated; otherwise, `NO`.

Discussion
`value` is set to `nil` if the requested resource value is not defined for the URL. In this case, the method still returns `YES`.

Availability
Available in iOS 4.0 and later.

See Also
"Common File System Resource Keys" (page 1399)
"File Property Keys" (page 1402)

"Volume Property Keys" (page 1403)

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 iOS 2.0 and later.

Declared In
NSURL.h

initByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:

Initializes a newly created NSURL that points to a location specified by resolving bookmark data.

```
- (id)initByResolvingBookmarkData:(NSData
    *)bookmarkDataoptions:(NSURLBookmarkResolutionOptions)optionsrelativeToURL:(NSURL
    *)relativeURLbookmarkDataIsStale:(BOOL *)isStaleerror:(NSError **)error
```

Parameters

bookmarkData
The bookmark data the URL is derived from.

options
Options taken into account when resolving the bookmark data.

relativeURL
The base URL that the bookmark data is relative to.

isStale
If YES, the bookmark data is stale.

error
The error that occurred in the case that the URL cannot be created.

Return Value
An NSURL initialized by resolving *bookmarkData*.

Availability
Available in iOS 4.0 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 1267).

Passing `nil` for this parameter produces an exception.

Return Value
An NSURL object initialized with *path*.

Discussion
Invoking this method is equivalent to invoking `initWithScheme:host:path:` (page 1388) 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 1387) which allows you to explicitly specify whether the returned NSURL represents a file or directory.

Availability
Available in iOS 2.0 and later.

See Also
`fileURLWithPath:` (page 1378)

Related Sample Code
AddMusic

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 1267).

Passing `nil` for this parameter produces an exception.

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 1388) with scheme `NSFileScheme`, a `nil` host, and *path*.

Availability
Available in iOS 2.0 and later.

See Also
`fileURLWithPath:` (page 1378)

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 the empty string.

path
The path for the NSURL object. If *path* begins with a tilde, it must first be expanded with `stringByExpandingTildeInPath` (page 1267).

Return Value
The newly initialized NSURL object.

Discussion
This method automatically escapes `path` with the `stringByAddingPercentEscapesUsingEncoding:` (page 1262) method.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`URLWithString:` (page 1381)

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 iOS 2.0 and later.

See Also
`- baseURL` (page 1383)
`- relativeString` (page 1393)
`URLWithString:relativeToURL:` (page 1381)

Declared In
`NSURL.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. Two NSURLs are considered equal if and only if they return identical values for both baseURL (page 1383) and relativeString (page 1393).

isFileReferenceURL

Returns whether the URL is a file reference URL.

```
- (BOOL)isFileReferenceURL
```

Return Value
YES if the URL is a file reference URL; otherwise, NO.

Availability
Available in iOS 4.0 and later.

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 iOS 2.0 and later.

Declared In
NSURL.h

lastPathComponent

Returns the last path component of a file URL.

```
- (NSString *)lastPathComponent
```

Return Value
The last path component of the URL.

Availability
Available in iOS 4.0 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 iOS 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 iOS 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, unescaped with the `stringByReplacingPercentEscapesUsingEncoding:` (page 1271) method. If the receiver does not conform to RFC 1808, returns `nil`. If this URL is a file URL (as determined with `isFileURL` (page 1390)), the return value is suitable for input into methods of NSFileManager or NSPathUtilities. If the path has a trailing slash it is stripped.

Availability
Available in iOS 2.0 and later.

Declared In
NSURL.h

pathComponents

Returns the individual path components of a file URL in an array.

```
- (NSArray *)pathComponents
```

Return Value
An array containing the individual path components of the URL.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

pathExtension

Returns the path extension of a file URL.

```
- (NSString *)pathExtension
```

Return Value
The path extension of the URL.

Availability
Available in iOS 4.0 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 iOS 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 iOS 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 1391). If the receiver does not conform to RFC 1808, returns `nil`.

Availability
Available in iOS 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 1382).

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
NSURL.h

resourceValuesForKeys:error:

Returns the resource values for the properties identified by specified array of keys.

```
- (NSDictionary *)resourceValuesForKeys:(NSArray *)keyserror:(NSError **)error
```

Parameters

keys
An array of names of URL resource properties.

error
The error that occurred in the case that one or more resource values cannot be retrieved.

Return Value
A dictionary of resource values indexed by key.

Discussion
If an error occurs, this method returns `nil`.

A key is left out of the returned dictionary if its corresponding resource value is not defined for the URL.

Availability
Available in iOS 4.0 and later.

See Also
"Common File System Resource Keys" (page 1399)
"File Property Keys" (page 1402)
"Volume Property Keys" (page 1403)

Declared In
NSURL.h

scheme

Returns the scheme of the URL.

```
- (NSString *)scheme
```

Return Value
The scheme of the URL.

Availability
Available in iOS 2.0 and later.

Declared In
NSURL.h

setResourceValue:forKey:error:

Sets the resource property of the URL specified by a given key to a given value.

```
- (BOOL)setResourceValue:(id)value forKey:(NSString *)key error:(NSError **)error
```

Parameters

value

The value for the resource property defined by *key*.

key

The name of one of the URL's resource properties.

error

The error that occurred in the case that the resource value cannot be set.

Return Value

`YES` if the resource property named *key* is successfully set to *value*; otherwise, `NO`.

Discussion

The resource is modified synchronously.

Availability

Available in iOS 4.0 and later.

See Also

"Common File System Resource Keys" (page 1399)

"File Property Keys" (page 1402)

"Volume Property Keys" (page 1403)

Declared In

`NSURL.h`

setResourceValues:error:

Sets resource properties of the URL specified by a given set of keys to a given set of values.

```
- (BOOL)setResourceValues:(NSDictionary *)keyedValues error:(NSError **)error
```

Parameters

keyedValues

A dictionary of resource values to be set.

error

The error that occurred in the case that one or more resource values cannot be set.

Return Value

`YES` if all resource values in *keyedValues* are successfully set; otherwise, `NO`.

Discussion

If an error occurs during the execution of this method, *error* will contain an array of the resource values that were not successfully set in its `userInfo` (page 434) dictionary.

Availability

Available in iOS 4.0 and later.

See Also

"Common File System Resource Keys" (page 1399)

"File Property Keys" (page 1402)

"Volume Property Keys" (page 1403)

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 iOS 2.0 and later.

Declared In
NSURL.h

URLByAppendingPathComponent:

Returns a new URL made by appending a path component to the original URL.

```
- (NSURL *)URLByAppendingPathComponent:(NSString *)pathComponent
```

Parameters
pathComponent
The path component to add to the URL.

Return Value
A new URL with pathComponent appended.

Discussion
If the original URL does not end with a forward slash and pathComponent does not begin with a forward slash, a forward slash is inserted between the two parts of the returned URL, unless the original URL is the empty string.

Availability
Available in iOS 4.0 and later.

Declared In
NSURL.h

URLByAppendingPathExtension:

Returns a new URL made by appending a path extension to the original URL.

```
- (NSURL *)URLByAppendingPathExtension:(NSString *)pathExtension
```

Parameters
pathExtension
The path extension to add to the URL.

Return Value
A new URL with `pathExtension` appended.

Discussion
If the original URL ends with one or more forward slashes, these are removed from the returned URL. A period is inserted between the two parts of the new URL.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

URLByDeletingLastPathComponent

Returns a new URL made by deleting the last path component from the original URL.

```
- (NSURL *)URLByDeletingLastPathComponent
```

Return Value
A new URL with the last path component of the original URL removed.

Discussion
If the original URL represents the root path, the returned URL is identical. Otherwise, if the original URL has only one path component, the new URL is the empty string.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

URLByDeletingPathExtension

Returns a new URL made by deleting the path extension, if any, from the original URL.

```
- (NSURL *)URLByDeletingPathExtension
```

Return Value
A new URL with the path extension of the original URL removed.

Discussion
If the original URL represents the root path, the returned URL is identical. If the URL has multiple path extensions, only the last one is removed.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

URLByResolvingSymlinksInPath

Returns a new URL that points to the same resource as the original URL and includes no symbolic links.

```
- (NSURL *)URLByResolvingSymlinksInPath
```

Return Value
A new URL that points to the same resource as the original URL and includes no symbolic links.

Discussion
If the original URL has no symbolic links, the returned URL is identical to the original URL.

This method only works on URLs with the `file:` path scheme. This method will return an identical URL for all other URLs.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURL.h`

URLByStandardizingPath

Returns a new URL that points to the same resource as the original URL and is an absolute path.

```
- (NSURL *)URLByStandardizingPath
```

Return Value
A new URL that points to the same resource as the original URL and is an absolute path.

Discussion
This method only works on URLs with the `file:` path scheme. This method will return an identical URL for all other URLs.

Availability
Available in iOS 4.0 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 iOS 2.0 and later.

Declared In
`NSURL.h`

Constants

NSURL Schemes

These schemes are the ones that `NSURL` can parse.

```
NSString * const NSURLFileScheme;
```

Constants

`NSURLFileScheme`

Identifies a URL that points to a file on a mounted volume.

Available in iOS 2.0 and later.

Declared in `NSURL.h`.

Discussion

For more information, see initWithScheme:host:path: (page 1388).

Common File System Resource Keys

Keys that are applicable to file system URLs.

```
NSString * const NSURLNameKey;
NSString * const NSURLLocalizedNameKey;
NSString * const NSURLIsRegularFileKey;
NSString * const NSURLIsDirectoryKey;
NSString * const NSURLIsSymbolicLinkKey;
NSString * const NSURLIsVolumeKey;
NSString * const NSURLIsPackageKey;
NSString * const NSURLIsSystemImmutableKey;
NSString * const NSURLIsUserImmutableKey;
NSString * const NSURLIsHiddenKey;
NSString * const NSURLHasHiddenExtensionKey;
NSString * const NSURLCreationDateKey;
NSString * const NSURLContentAccessDateKey;
NSString * const NSURLContentModificationDateKey;
NSString * const NSURLAttributeModificationDateKey;
NSString * const NSURLLinkCountKey;
NSString * const NSURLParentDirectoryURLKey;
NSString * const NSURLVolumeURLKey;
NSString * const NSURLTypeIdentifierKey;
NSString * const NSURLLocalizedTypeDescriptionKey;
NSString * const NSURLLabelNumberKey;
NSString * const NSURLLabelColorKey;
NSString * const NSURLLocalizedLabelKey;
NSString * const NSURLEffectiveIconKey;
NSString * const NSURLCustomIconKey;
```

Constants

`NSURLNameKey`

Key for the resource's name in the file system, returned as an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLocalizedNameKey`
Key for the resource's localized or extension-hidden name, retuned as an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsRegularFileKey`
Key for determining whether the resource is a regular file, as opposed to a directory or a symbolic link. Returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsDirectoryKey`
Key for determining whether the resource is a directory, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsSymbolicLinkKey`
Key for determining whether the resource is a symbolic link, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsVolumeKey`
Key for determining whether the resource is the root directory of a volume, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsPackageKey`
Key for determining whether the resource is a packaged directory, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsSystemImmutableKey`
Key for determining whether the resource's system immutable bit is set, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsUserImmutableKey`
Key for determining whether the resource's user immutable bit is set, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsHiddenKey`
Key for determining whether the resource is normally not displayed to users, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLHasHiddenExtensionKey`
Key for determining whether the resource's extension is normally removed from its localized name, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLCreationDateKey`
Key for the resource's creation date, returned as an `NSDate` object if the volume supports creation dates, or `nil` if creation dates are unsupported.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLContentAccessDateKey`
Key for the last time the resource was accessed, returned as an `NSDate` object if the volume supports access dates, or `nil` if access dates are unsupported.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLContentModificationDateKey`
Key for the last time the resource was modified, returned as an `NSDate` object if the volume supports modification dates, or `nil` if modification dates are unsupported.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLAttributeModificationDateKey`
Key for the last time the resource's attributes were modified, returned as an `NSDate` object if the volume supports attribute modification dates, or `nil` if attribute modification dates are unsupported.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLinkCountKey`
Key for the number of hard links to the resource, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLParentDirectoryURLKey`
Key for the parent directory of the resource, returned as an `NSURL` object, or `nil` if the resource is the root directory of its volume.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeURLKey`
Key for the root directory of the resource's volume, returned as an `NSURL` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLTypeIdentifierKey`
Key for the resource's uniform type identifier (UTI), returned as an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLocalizedTypeDescriptionKey`
Key for the resource's localized type description, returned as an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLabelNumberKey`
Key for the resource's label number, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLabelColorKey`
Key for the resource's label color, returned as an `NSColor` object, or `nil` if the resource has no label color.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLLocalizedLabelKey`
Key for the resource's localized label text, returned as an `NSString` object, or `nil` if the resource has no localized label text.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLEffectiveIconKey`
Key for the resource's normal icon, returned as an `NSImage` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLCustomIconKey`
Key for the icon stored with the resource, returned as an `NSImage` object, or `nil` if the resource has no custom icon.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

File Property Keys

Keys that apply to properties of files.

```
NSString * const NSURLFileSizeKey;
NSString * const NSURLFileAllocatedSizeKey;
NSString * const NSURLIsAliasFileKey;
```

Constants

`NSURLFileSizeKey`
Key for the file's size in bytes, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLFileAllocatedSizeKey`
Key for the total size allocated on disk for the file, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLIsAliasFileKey`
Key for determining whether the file is an alias, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

Volume Property Keys

Keys that apply to volumes.

```
NSString * const NSURLVolumeLocalizedFormatDescriptionKey;
NSString * const NSURLVolumeTotalCapacityKey;
NSString * const NSURLVolumeAvailableCapacityKey;
NSString * const NSURLVolumeResourceCountKey;
NSString * const NSURLVolumeSupportsPersistentIDsKey;
NSString * const NSURLVolumeSupportsSymbolicLinksKey;
NSString * const NSURLVolumeSupportsHardLinksKey;
NSString * const NSURLVolumeSupportsJournalingKey;
NSString * const NSURLVolumeIsJournalingKey;
NSString * const NSURLVolumeSupportsSparseFilesKey;
NSString * const NSURLVolumeSupportsZeroRunsKey;
NSString * const NSURLVolumeSupportsCaseSensitiveNamesKey;
NSString * const NSURLVolumeSupportsCasePreservedNamesKey;
```

Constants

`NSURLVolumeLocalizedFormatDescriptionKey`
Key for the volume's descriptive format name, returned as an `NSString` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeTotalCapacityKey`
Key for the volume's capacity in bytes, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeAvailableCapacityKey`
Key for the volume's available capacity in bytes, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeResourceCountKey`
Key for the total number of resources on the volume, returned as an `NSNumber` object.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsPersistentIDsKey`
Key for determining whether the volume supports persistent IDs, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsSymbolicLinksKey`
Key for determining whether the volume supports symbolic links, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsHardLinksKey`
Key for determining whether the volume supports hard links, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsJournalingKey`
Key for determining whether the volume supports journaling, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeIsJournalingKey`
Key for determining whether the volume is currently journaling, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsSparseFilesKey`
Key for determining whether the volume supports sparse files, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsZeroRunsKey`
Key for determining whether the volume supports zero runs, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsCaseSensitiveNamesKey`
Key for determining whether the volume supports case-sensitive names, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLVolumeSupportsCasePreservedNamesKey`
Key for determining whether the volume supports case-preserved names, returned as an `NSNumber` object with value `0` or `1`.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

Bookmark Data Creation Options

Options used when creating bookmark data.

```
enum {
    NSURLBookmarkCreationPreferFileIDResolution = ( 1UL << 8 ),
    NSURLBookmarkCreationMinimalBookmark = ( 1UL << 9 ),
    NSURLBookmarkCreationSuitableForBookmarkFile = ( 1UL << 10 )
};
typedef NSUInteger NSURLBookmarkCreationOptions;
typedef NSUInteger NSURLBookmarkFileCreationOptions;
```

Constants

`NSURLBookmarkCreationPreferFileIDResolution`

Option for specifying that an alias created with the bookmark data prefers resolving with its embedded file ID.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLBookmarkCreationMinimalBookmark`

Option for specifying that an alias created with the bookmark data be created with minimal information, which may make it smaller but still able to resolve in certain ways.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLBookmarkCreationSuitableForBookmarkFile`

Option for specifying that the bookmark data include properties required to create Finder alias files.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

Bookmark Data Resolution Options

Options used when resolving bookmark data.

```
enum {
    NSURLBookmarkResolutionWithoutUI = ( 1UL << 8 ),
    NSURLBookmarkResolutionWithoutMounting = ( 1UL << 9 )
};
typedef NSUInteger NSURLBookmarkResolutionOptions;
```

Constants

`NSURLBookmarkResolutionWithoutUI`

Option for specifying that no UI feedback accompany resolution of the bookmark data.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

`NSURLBookmarkResolutionWithoutMounting`

Option for specifying that no volume should be mounted during resolution of the bookmark data.

Available in iOS 4.0 and later.

Declared in `NSURL.h`.

NSURLAuthenticationChallenge Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLAuthenticationChallenge.h
Companion guide	URL Loading System Programming Guide

Overview

NSURLAuthenticationChallenge encapsulates a challenge from a server requiring authentication from the client.

Tasks

Creating an Authentication Challenge Instance

`- initWithAuthenticationChallenge:sender:` (page 1409)
Returns an initialized NSURLAuthenticationChallenge object copying the properties from *challenge*, and setting the authentication sender to *sender*.

`- initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender:` (page 1409)
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 1408)
Returns the NSError object representing the last authentication failure.

`- failureResponse` (page 1408)
Returns the NSURLResponse object representing the last authentication failure.

- `previousFailureCount` (page 1409)
  Returns the receiver's count of failed authentication attempts.
- `proposedCredential` (page 1410)
  Returns the proposed credential for this challenge.
- `protectionSpace` (page 1410)
  Returns the receiver's protection space.
- `sender` (page 1410)
  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

See Also
- `failureResponse` (page 1408)

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

See Also
- `error` (page 1408)

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

See Also
- initWithProtectionSpace:proposedCredential:previousFailureCount:failureResponse:error:sender: (page 1409)

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

See Also
- initWithAuthenticationChallenge:sender: (page 1409)

Declared In
NSURLAuthenticationChallenge.h

previousFailureCount

Returns the receiver's count of failed authentication attempts.

```
- (NSInteger)previousFailureCount
```

Availability

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 you have previously attempted to authenticate and failed, this method returns the most recent failed credential.

If the proposed credential is not `nil` and returns `YES` when sent the message `hasPassword` (page 1438), 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

Declared In
`NSURLAuthenticationChallenge.h`

protectionSpace

Returns the receiver's protection space.

```
- (NSURLProtectionSpace *)protectionSpace
```

Availability

Declared In
`NSURLAuthenticationChallenge.h`

sender

Returns the receiver's sender.

```
- (id < NSURLAuthenticationChallengeSender >)sender
```

Discussion
The sender should be sent a `useCredential:forAuthenticationChallenge:` (page 1646), `continueWithoutCredentialForAuthenticationChallenge:` (page 1646) or `cancelAuthenticationChallenge:` (page 1646) when the client is finished processing the authentication challenge.

Availability

Declared In
`NSURLAuthenticationChallenge.h`

NSURLCache Class Reference

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

Overview

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 1413)
Returns the shared NSURLCache instance.

`+ setSharedURLCache:` (page 1412)
Sets the shared NSURLCache instance to a specified cache object.

Creating a New Cache Object

`- initWithMemoryCapacity:diskCapacity:diskPath:` (page 1415)
Initializes an NSURLCache object with the specified values.

Getting and Storing Cached Objects

- `cachedResponseForRequest:` (page 1413)
  Returns the cached URL response in the cache for the specified URL request.
- `storeCachedResponse:forRequest:` (page 1417)
  Stores a cached URL response for a specified request

Removing Cached Objects

- `removeAllCachedResponses` (page 1416)
  Clears the receiver's cache, removing all stored cached URL responses.
- `removeCachedResponseForRequest:` (page 1416)
  Removes the cached URL response for a specified URL request.

Getting and Setting On-disk Cache Properties

- `currentDiskUsage` (page 1414)
  Returns the current size of the receiver's on-disk cache, in bytes.
- `diskCapacity` (page 1414)
  Returns the capacity of the receiver's on-disk cache, in bytes.
- `setDiskCapacity:` (page 1416)
  Sets the receiver's on-disk cache capacity

Getting and Setting In-memory Cache Properties

- `currentMemoryUsage` (page 1414)
  Returns the current size of the receiver's in-memory cache, in bytes.
- `memoryCapacity` (page 1415)
  Returns the capacity of the receiver's in-memory cache, in bytes.
- `setMemoryCapacity:` (page 1417)
  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
An application that has special caching requirements or constraints should use this method to specify an NSURLCache instance with customized cache settings. The application should do so before any calls to the `sharedURLCache` (page 1413) method.

Availability

See Also
`+ sharedURLCache` (page 1413)

Declared In
`NSURLCache.h`

sharedURLCache

Returns the shared NSURLCache instance.

```
+ (NSURLCache *)sharedURLCache
```

Return Value
The shared NSURLCache instance.

Discussion
Applications that do not have special caching requirements or constraints should find the default shared cache instance acceptable. An application with more specific needs can create a custom `NSURLCache` object and set it as the shared cache instance using `setSharedURLCache:` (page 1412). The application should do so before any calls to this method.

Availability

See Also
`+ setSharedURLCache:` (page 1412)

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

See Also
- `storeCachedResponse:forRequest:` (page 1417)

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

See Also
- `diskCapacity` (page 1414)
- `setDiskCapacity:` (page 1416)

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

See Also
- `memoryCapacity` (page 1415)
- `setMemoryCapacity:` (page 1417)

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

See Also
- `currentDiskUsage` (page 1414)
- `setDiskCapacity:` (page 1416)

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
In Mac OS X, *path* is the location at which to store the on-disk cache.

In iOS, *path* is the name of a subdirectory of the application's default cache directory in which to store the on-disk cache (the subdirectory is created if it does not exist).

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

See Also
+ `sharedURLCache` (page 1413)

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

See Also
- `currentMemoryUsage` (page 1414)
- `setMemoryCapacity:` (page 1417)

Declared In
`NSURLCache.h`

removeAllCachedResponses

Clears the receiver's cache, removing all stored cached URL responses.

```
- (void)removeAllCachedResponses
```

Availability

See Also
- `removeCachedResponseForRequest:` (page 1416)

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

See Also
- `removeAllCachedResponses` (page 1416)

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

See Also
- `currentDiskUsage` (page 1414)
- `diskCapacity` (page 1414)

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

See Also
- `currentMemoryUsage` (page 1414)
- `memoryCapacity` (page 1415)

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

See Also
- `cachedResponseForRequest:` (page 1413)

Declared In
`NSURLCache.h`

NSURLConnection Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLConnection.h
Companion guide	URL Loading System Programming Guide

Overview

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.

NSURLConnection retains its delegate when it is initialized. It releases the delegate when the connection finishes loading, fails, or is canceled.

The following contract governs the delegate methods defined in this interface:

- Zero or more `connection:willSendRequest:redirectResponse:` (page 1431) 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 1428) 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 1427) 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 1429) messages will be sent to the delegate before receiving a `connection:didReceiveData:` (page 1429) 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 1429) messages will be sent before any of the following messages are sent to the delegate: `connection:willCacheResponse:` (page 1431), `connectionDidFinishLoading:` (page 1432), `connection:didFailWithError:` (page 1428).
- Zero or one `connection:willCacheResponse:` (page 1431) messages will be sent to the delegate after `connection:didReceiveData:` (page 1429) is sent but before a `connectionDidFinishLoading:` (page 1432) message is sent.
- Unless a NSURLConnection receives a `cancel` (page 1424) message, the delegate will receive one and only one of `connectionDidFinishLoading:` (page 1432), or `connection:didFailWithError:` (page 1428) 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 1423), 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 1422)
Returns whether a request can be handled based on a "preflight" evaluation.

Loading Data Synchronously

`+ sendSynchronousRequest:returningResponse:error:` (page 1423)
Performs a synchronous load of the specified URL request.

Loading Data Asynchronously

`+ connectionWithRequest:delegate:` (page 1422)
Creates and returns an initialized URL connection and begins to load the data for the URL request.

`- initWithRequest:delegate:` (page 1424)
Returns an initialized URL connection and begins to load the data for the URL request.

`- initWithRequest:delegate:startImmediately:` (page 1425)
Returns an initialized URL connection and begins to load the data for the URL request, if specified.

`- start` (page 1426)
Causes the receiver to begin loading data, if it has not already.

Stopping a Connection

- `cancel` (page 1424)
  Cancels an asynchronous load of a request.

Runloop Scheduling

- `scheduleInRunLoop:forMode:` (page 1425)
  Determines the runloop and mode that the receiver uses to send delegate messages to the receiver.
- `unscheduleFromRunLoop:forMode:` (page 1426)
  Causes the receiver to stop sending delegate messages using the specified runloop and mode.

Connection Authentication

- `connection:canAuthenticateAgainstProtectionSpace:` (page 1427) *delegate method*
  Sent to determine whether the delegate is able to respond to a protection space's form of authentication.
- `connection:didCancelAuthenticationChallenge:` (page 1427) *delegate method*
  Sent when a connection cancels an authentication challenge.
- `connection:didReceiveAuthenticationChallenge:` (page 1428) *delegate method*
  Sent when a connection must authenticate a challenge in order to download its request.
- `connectionShouldUseCredentialStorage:` (page 1432) *delegate method*
  Sent to determine whether the URL loader should consult the credential storage for authenticating the connection.

Connection Data and Responses

- `connection:willCacheResponse:` (page 1431) *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 1429) *delegate method*
  Sent when the connection has received sufficient data to construct the URL response for its request.
- `connection:didReceiveData:` (page 1429) *delegate method*
  Sent as a connection loads data incrementally.
- `connection:didSendBodyData:totalBytesWritten:totalBytesExpectedToWrite:` (page 1430) *delegate method*
  Sent as the body (message data) of a request is transmitted (such as in an http POST request).
- `connection:willSendRequest:redirectResponse:` (page 1431) *delegate method*
  Sent when the connection determines that it must change URLs in order to continue loading a request.

Connection Completion

`- connection:didFailWithError:` (page 1428) *delegate method*
Sent when a connection fails to load its request successfully.

`- connectionDidFinishLoading:` (page 1432) *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

See Also
`+ registerClass:` (page 1462)
`+ unregisterClass:` (page 1465)

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

See Also
`- initWithRequest:delegate:` (page 1424)

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

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

See Also
`+ connectionWithRequest:delegate:` (page 1422)
`- initWithRequest:delegate:` (page 1424)

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 1425) to change the runloop and mode.

Return Value
The URL connection for the URL request. Returns `nil` if a connection can't be initialized.

Special Considerations

The connection retains *delegate*. It releases *delegate* when the connection finishes loading, fails, or is canceled.

Availability

See Also
`+ connectionWithRequest:delegate:` (page 1422)
`- initWithRequest:delegate:startImmediately:` (page 1425)

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 1425) to change the runloop and mode.]

startImmediately

`YES` if the connection should being loading data immediately, otherwise `NO`. If you pass `NO`, you must schedule the connection in a run loop before starting it.

Return Value
The URL connection for the URL request. Returns `nil` if a connection can't be initialized.

Special Considerations

The connection retains *delegate*. It releases *delegate* when the connection finishes loading, fails, or is canceled.

Availability
Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared In
NSURLConnection.h

Delegate Methods

connection:canAuthenticateAgainstProtectionSpace:

Sent to determine whether the delegate is able to respond to a protection space's form of authentication.

```
- (BOOL)connection:(NSURLConnection *)connection
    canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)protectionSpace
```

Parameters

connection

The connection sending the message.

protectionSpace

The protection space that generates an authentication challenge.

Discussion
This method is called before `connection:didReceiveAuthenticationChallenge:` (page 1428), allowing the delegate to inspect a protection space before attempting to authenticate against it. By returning `YES`, the delegate indicates that it can handle the form of authentication, which it does in the subsequent call to `connection:didReceiveAuthenticationChallenge:` (page 1428). If the delegate returns `NO`, the system attempts to use the user's keychain to authenticate. If your delegate does not implement this method and the protection space uses client certificate authentication or server trust authentication, the system behaves as if you returned `NO`. The system behaves as if you returned `YES` for all other authentication methods.

Availability
Available in iOS 3.0 and later.

Declared In
NSURLConnection.h

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

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

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 1409) to *challenge*.

If the previous failure count is 0 and the value returned by `proposedCredential` (page 1410) is `nil`, the delegate can create a new NSURLCredential object, providing information specific to the type of credential, and send a `useCredential:forAuthenticationChallenge:` (page 1646) 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 1646) or a `cancelAuthenticationChallenge:` (page 1646) message. The specific action is 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 1646), `continueWithoutCredentialForAuthenticationChallenge:` (page 1646) or `cancelAuthenticationChallenge:` (page 1646).

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 1646) with the credential. If the challenge has no credential or the credentials fail to authorize access, then `continueWithoutCredentialForAuthenticationChallenge:` (page 1646) is sent to `[challenge sender]` instead.

Availability

See Also
- `cancelAuthenticationChallenge:` (page 1646)
- `continueWithoutCredentialForAuthenticationChallenge:` (page 1646)
- `useCredential:forAuthenticationChallenge:` (page 1646)

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

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

Declared In

`NSURLConnection.h`

connection:didSendBodyData:totalBytesWritten:totalBytesExpectedToWrite:

Sent as the body (message data) of a request is transmitted (such as in an http POST request).

```
- (void)connection:(NSURLConnection *)connection
    didSendBodyData:(NSInteger)bytesWritten
    totalBytesWritten:(NSInteger)totalBytesWritten
    totalBytesExpectedToWrite:(NSInteger)totalBytesExpectedToWrite
```

Parameters

connection

The connection sending the message.

bytesWritten

The number of bytes written in the latest write.

totalBytesWritten

The total number of bytes written for this connection.

totalBytesExpectedToWrite

The number of bytes the connection expects to write.

Discussion

This method provides an estimate of the progress of a URL upload.

The value of `totalBytesExpectedToWrite` may change during the upload if the request needs to be retransmitted due to a lost connection or an authentication challenge from the server.

Availability

Available in iOS 3.0 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

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 return *request* unmodified to allow the redirect, return a new request, or return nil to reject the redirect and continue processing the connection.

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.

The delegate can receive this message as a result of modifying a request before it is sent, for example to transform the request's URL to its canonical form. To detect this case, examine *redirectResponse*; if it is `nil`, the message was not sent due to a redirect.

The delegate should be prepared to receive this message multiple times.

Availability

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

Declared In
`NSURLConnection.h`

connectionShouldUseCredentialStorage:

Sent to determine whether the URL loader should consult the credential storage for authenticating the connection.

```
- (BOOL)connectionShouldUseCredentialStorage:(NSURLConnection *)connection
```

Parameters

connection
The connection sending the message.

Discussion
This method is called before any attempt to authenticate is made. By returning `NO`, the delegate tells the connection not to consult the credential storage and makes itself responsible for providing credentials for any authentication challenges. Not implementing this method is the same as returning `YES`. The delegate is free to consult the credential storage itself when it receives a `connection:didReceiveAuthenticationChallenge:` (page 1428) message.

Availability
Available in iOS 3.0 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 iOS 2.0 and later.
Declared in	Foundation/NSURLCredential.h
Companion guide	URL Loading System Programming Guide

Overview

`NSURLCredential` is an immutable object representing an authentication credential consisting of authentication information specific to the type of credential and the type of persistent storage to use, if any.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 1554)

Tasks

Creating a Credential

`+ credentialForTrust:` (page 1436)
Creates and returns an `NSURLCredential` object for server trust authentication with a given accepted trust.

`+ credentialWithUser:password:persistence:` (page 1437)
Creates and returns an `NSURLCredential` object for internet password authentication with a given user name and password using a given persistence setting.

+ `credentialWithIdentity:certificates:persistence:` (page 1437)
  Creates and returns an `NSURLCredential` object for client certificate authentication with a given identity and a given array of client certificates using a given persistence setting.

- `initWithIdentity:certificates:persistence:` (page 1439)
  Returns an `NSURLCredential` object for client certificate authentication initialized with a given identity and a given array of client certificates using a given persistence setting.

- `initWithTrust:` (page 1439)
  Returns an `NSURLCredential` object for server trust authentication initialized with a given accepted trust.

- `initWithUser:password:persistence:` (page 1440)
  Returns an `NSURLCredential` object initialized with a given user name and password using a given persistence setting.

Getting Credential Properties

- `certificates` (page 1438)
  Returns an array of `SecCertificateRef` objects representing the certificates of the credential if it is a client certificate credential.

- `hasPassword` (page 1438)
  Returns a Boolean value that indicates whether the receiver has a password.

- `identity` (page 1439)
  Returns the identity of this credential if it is a client certificate credential.

- `password` (page 1440)
  Returns the receiver's password.

- `persistence` (page 1441)
  Returns the receiver's persistence setting.

- `user` (page 1441)
  Returns the receiver's user name.

Class Methods

credentialForTrust:

Creates and returns an `NSURLCredential` object for server trust authentication with a given accepted trust.

```
+ (NSURLCredential *)credentialForTrust:(SecTrustRef)trust
```

Parameters

trust
  The accepted trust.

Discussion
Before creating a server trust credential, it is the responsibility of the delegate of an `NSURLConnection` object or an `NSURLDownload` object to evaluate the trust. Do this by calling `SecTrustEvaluate`, passing it the trust obtained from the `serverTrust` method of the server's `NSURLProtectionSpace` object. If the trust is invalid, the authentication challenge should be cancelled with `cancelAuthenticationChallenge:` (page 1646).

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLCredential.h`

credentialWithIdentity:certificates:persistence:

Creates and returns an `NSURLCredential` object for client certificate authentication with a given identity and a given array of client certificates using a given persistence setting.

```
+ (NSURLCredential *)credentialWithIdentity:(SecIdentityRef)identity
    certificates:(NSArray *)certArray
    persistence:(NSURLCredentialPersistence)persistence
```

Parameters

identity
The identity for the credential.

certArray
An array of one or more `SecCertificateRef` objects representing certificates for the credential.

persistence
The persistence setting for the credential.

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLCredential.h`

credentialWithUser:password:persistence:

Creates and returns an `NSURLCredential` object for internet password authentication 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

See Also
`- initWithUser:password:persistence:` (page 1440)

Declared In
`NSURLCredential.h`

Instance Methods

certificates

Returns an array of `SecCertificateRef` objects representing the certificates of the credential if it is a client certificate credential.

```
- (NSArray *)certificates
```

Return Value
The certificates of the credential, or `nil` if this is not a client certificate credential.

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLCredential.h`

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 1440) may return `NO` even if this method returns `YES`, since getting the password may fail, or the user may refuse access.

Availability

Declared In
`NSURLCredential.h`

identity

Returns the identity of this credential if it is a client certificate credential.

```
- (SecIdentityRef)identity
```

Return Value
The identity of the credential, or `NULL` if this is not a client certificate credential.

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLCredential.h`

initWithIdentity:certificates:persistence:

Returns an `NSURLCredential` object for client certificate authentication initialized with a given identity and a given array of client certificates using a given persistence setting.

```
- (id)initWithIdentity:(SecIdentityRef)identity certificates:(NSArray *)certArray
    persistence:(NSURLCredentialPersistence)persistence
```

Parameters

identity
The identity for the credential.

certArray
An array of one or more `SecCertificateRef` objects representing certificates for the credential.

persistence
The persistence setting for the credential.

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLCredential.h`

initWithTrust:

Returns an `NSURLCredential` object for server trust authentication initialized with a given accepted trust.

```
- (id)initWithTrust:(SecTrustRef)trust
```

Parameters

trust
The accepted trust.

Discussion
Before creating a server trust credential, it is the responsibility of the delegate of an `NSURLConnection` object or an `NSURLDownload` object to evaluate the trust. Do this by calling `SecTrustEvaluate`, passing it the trust obtained from the `serverTrust` method of the server's `NSURLProtectionSpace` object. If the trust is invalid, the authentication challenge should be cancelled with `cancelAuthenticationChallenge:` (page 1646).

Availability
Available in iOS 3.0 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

See Also
+ credentialWithUser:password:persistence: (page 1437)

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

See Also
- hasPassword (page 1438)

Declared In
NSURLCredential.h

persistence

Returns the receiver's persistence setting.

```
- (NSURLCredentialPersistence)persistence
```

Return Value
The receiver's persistence setting.

Availability

Declared In
NSURLCredential.h

user

Returns the receiver's user name.

```
- (NSString *)user
```

Return Value
The receiver's user name.

Availability

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 iOS 2.0 and later.

Declared in NSURLCredential.h.

`NSURLCredentialPersistenceForSession`
Credential will be stored only for this session.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSURLCredential.h`.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSURLCredentialStorage.h
Companion guide	URL Loading System Programming Guide

Overview

NSURLCredentialStorage implements a singleton (shared object) that manages the credential storage.

Tasks

Getting the Credential Storage

`+ sharedCredentialStorage` (page 1444)
Returns the shared URL credential storage object.

Getting and Setting Default Credentials

`- defaultCredentialForProtectionSpace:` (page 1445)
Returns the default credential for the specified *protectionSpace*.

`- setDefaultCredential:forProtectionSpace:` (page 1446)
Sets the default credential for a specified protection space.

Adding and Removing Credentials

`- removeCredential:forProtectionSpace:` (page 1445)
Removes a specified credential from the credential storage for the specified protection space.

- `setCredential:forProtectionSpace:` (page 1446)
  Adds *credential* to the credential storage for the specified *protectionSpace*.

Retrieving Credentials

- `allCredentials` (page 1444)
  Returns a dictionary containing the credentials for all available protection spaces.
- `credentialsForProtectionSpace:` (page 1445)
  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

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

See Also
`- credentialsForProtectionSpace:` (page 1445)

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

See Also

`- allCredentials` (page 1444)

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

See Also

`- setDefaultCredential:forProtectionSpace:` (page 1446)

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

See Also
- setCredential:forProtectionSpace: (page 1446)

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

See Also
- removeCredential:forProtectionSpace: (page 1445)

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

See Also
- defaultCredentialForProtectionSpace: (page 1445)

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

Declared In
`NSURLCredentialStorage.h`

NSURLProtectionSpace Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLProtectionSpace.h
Companion guide	URL Loading System Programming Guide

Overview

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 1554)

Tasks

Creating a Protection Space

- `initWithHost:port:protocol:realm:authenticationMethod:` (page 1451)
  Initializes a protection space object.
- `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1452)
  Initializes a protection space object representing a proxy server.

Getting Protection Space Properties

- `authenticationMethod` (page 1450)
  Returns the authentication method used by the receiver.
- `distinguishedNames` (page 1450)
  Returns an array of acceptable certificate-issuing authorities for client certificate authentication.
- `host` (page 1451)
  Returns the receiver's host.
- `isProxy` (page 1452)
  Returns whether the receiver represents a proxy server.
- `port` (page 1453)
  Returns the receiver's port.
- `protocol` (page 1453)
  Returns the receiver's protocol.
- `proxyType` (page 1453)
  Returns the receiver's proxy type.
- `realm` (page 1453)
  Returns the receiver's authentication realm
- `receivesCredentialSecurely` (page 1454)
  Returns whether the credentials for the protection space can be sent securely.
- `serverTrust` (page 1454)
  Returns a representation of the server's SSL transaction state.

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 1454).

Availability

Declared In
`NSURLProtectionSpace.h`

distinguishedNames

Returns an array of acceptable certificate-issuing authorities for client certificate authentication.

```
- (NSArray *)distinguishedNames
```

Return Value
An array of acceptable certificate-issuing authorities, or `nil` if the authentication method of the protection space is not client certificate.

Discussion
The returned issuing authorities are encoded with Distinguished Encoding Rules (DER).

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLProtectionSpace.h`

host

Returns the receiver's host.

```
- (NSString *)host
```

Return Value
The receiver's host.

Availability

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 1454) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability

See Also
- `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1452)

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 1454).

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 1454) or `nil` to use the default, `NSURLAuthenticationMethodDefault`.

Availability

See Also
- `initWithHost:port:protocol:realm:authenticationMethod:` (page 1451)

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

Declared In
NSURLProtectionSpace.h

port

Returns the receiver's port.

```
- (NSInteger)port
```

Return Value
The receiver's port.

Availability

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

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 1454).

Availability

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

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

Declared In
`NSURLProtectionSpace.h`

serverTrust

Returns a representation of the server's SSL transaction state.

```
- (SecTrustRef)serverTrust
```

Return Value
The server's SSL transaction state, or `nil` if the authentication method of the protection space is not server trust.

Availability
Available in iOS 3.0 and later.

Declared In
`NSURLProtectionSpace.h`

Constants

NSURLProtectionSpace Protocol Types

These constants describe the supported protocols for a protection space, as returned by `protocol` (page 1453).

```
NSString * const NSURLProtectionSpaceHTTP;
NSString * const NSURLProtectionSpaceHTTPS;
NSString * const NSURLProtectionSpaceFTP;
```

Constants

`NSURLProtectionSpaceHTTP`

The protocol type for HTTP.

Available in iOS 4.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceHTTPS`

The protocol type for HTTPS.

Available in iOS 4.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceFTP`

The protocol type for FTP.

Available in iOS 4.0 and later.

Declared in `NSURLProtectionSpace.h`.

NSURLProtectionSpace Proxy Types

These constants describe the supported proxy types used in `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1452) and returned by `proxyType` (page 1453).

```
NSString *NSURLProtectionSpaceHTTPProxy;
NSString *NSURLProtectionSpaceHTTPSProxy;
NSString *NSURLProtectionSpaceFTPProxy;
NSString *NSURLProtectionSpaceSOCKSProxy;
```

Constants

`NSURLProtectionSpaceHTTPProxy`

The proxy type for HTTP proxies.

Available in iOS 2.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceHTTPSProxy`

The proxy type for HTTPS proxies.

Available in iOS 2.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceFTPProxy`

The proxy type for FTP proxies.

Available in iOS 2.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLProtectionSpaceSOCKSProxy`

The proxy type for SOCKS proxies.

Available in iOS 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 1451), `initWithProxyHost:port:type:realm:authenticationMethod:` (page 1452) and returned by `authenticationMethod` (page 1450).

```
NSString *NSURLAuthenticationMethodDefault;
NSString *NSURLAuthenticationMethodHTTPBasic;
NSString *NSURLAuthenticationMethodHTTPDigest;
NSString *NSURLAuthenticationMethodHTMLForm;
NSString *NSURLAuthenticationMethodNegotiate;
NSString *NSURLAuthenticationMethodClientCertificate;
NSString *NSURLAuthenticationMethodServerTrust;
```

Constants

`NSURLAuthenticationMethodDefault`
Use the default authentication method for a protocol.
Available in iOS 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 iOS 2.0 and later.
Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodHTTPDigest`
Use HTTP digest authentication for this protection space.
Available in iOS 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 iOS 2.0 and later.
Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodNegotiate`
Use negotiate authentication for this protection space.
Available in iOS 4.0 and later.
Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodClientCertificate`

Use client certificate authentication for this protection space.

This authentication method can apply to any protocol.

Available in iOS 3.0 and later.

Declared in `NSURLProtectionSpace.h`.

`NSURLAuthenticationMethodServerTrust`

Use server trust authentication for this protection space.

This authentication method can apply to any protocol.

Available in iOS 3.0 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 iOS 2.0 and later.
Declared in	Foundation/NSURLProtocol.h
Companion guide	URL Loading System Programming Guide

Overview

`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 1462) and `setProperty:forKey:inRequest:` (page 1464).

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 1466)
Initializes an `NSURLProtocol` object.

Registering and Unregistering Protocol Classes

`+ registerClass:` (page 1462)
Attempts to register a subclass of `NSURLProtocol`, making it visible to the URL loading system.

`+ unregisterClass:` (page 1465)
Unregisters the specified subclass of `NSURLProtocol`.

Getting and Setting Request Properties

`+ propertyForKey:inRequest:` (page 1462)
Returns the property associated with the specified key in the specified request.

`+ setProperty:forKey:inRequest:` (page 1464)
Sets the property associated with the specified key in the specified request.

`+ removePropertyForKey:inRequest:` (page 1463)
Removes the property associated with the specified key in the specified request.

Determining If a Subclass Can Handle a Request

`+ canInitWithRequest:` (page 1461)
Returns whether the protocol subclass can handle the specified request.

Providing a Canonical Version of a Request

`+ canonicalRequestForRequest:` (page 1461)
Returns a canonical version of the specified request.

Determining If Requests Are Cache Equivalent

`+ requestIsCacheEquivalent:toRequest:` (page 1464)
Returns whether two requests are equivalent for cache purposes.

Starting and Stopping Downloads

- `startLoading` (page 1466)
  Starts protocol-specific loading of the request.
- `stopLoading` (page 1467)
  Stops protocol-specific loading of the request.

Getting Protocol Attributes

- `cachedResponse` (page 1465)
  Returns the receiver's cached response.
- `client` (page 1465)
  Returns the object the receiver uses to communicate with the URL loading system.
- `request` (page 1466)
  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

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

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

See Also

`+ setProperty:forKey:inRequest:` (page 1464)

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 1461) 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 1461).

Availability

See Also

+ unregisterClass: (page 1465)

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 iOS 2.0 and later.

See Also

+ propertyForKey:inRequest: (page 1462)

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

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

See Also

`+ propertyForKey:inRequest:` (page 1462)

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

See Also
+ registerClass: (page 1462)

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

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

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

Declared In
`NSURLProtocol.h`

request

Returns the receiver's request.

```
- (NSURLRequest *)request
```

Return Value
The receiver's request.

Availability

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

See Also
- `stopLoading` (page 1467)

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

See Also
- `startLoading` (page 1466)

Declared In
`NSURLProtocol.h`

NSURLRequest Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLRequest.h
Companion guide	URL Loading System Programming Guide

Overview

`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 1462) and `setProperty:forKey:inRequest:` (page 1464) methods.

The mutable subclass of NSURLRequest is NSMutableURLRequest.

Adopted Protocols

NSCopying
- `copyWithZone:` (page 1554)

NSMutableCopying
- `mutableCopyWithZone:` (page 1614)

Tasks

Creating Requests

+ `requestWithURL:` (page 1471)
  Creates and returns a URL request for a specified URL with default cache policy and timeout value.

- `initWithURL:` (page 1474)
  Returns a URL request for a specified URL with default cache policy and timeout value.

+ `requestWithURL:cachePolicy:timeoutInterval:` (page 1471)
  Creates and returns an initialized URL request with specified values.

- `initWithURL:cachePolicy:timeoutInterval:` (page 1475)
  Returns an initialized URL request with specified values.

Getting Request Properties

- `cachePolicy` (page 1472)
  Returns the receiver's cache policy.

- `HTTPShouldUsePipelining` (page 1474)
  Returns whether the request should continue transmitting data before receiving a response from an earlier transmission.

- `mainDocumentURL` (page 1475)
  Returns the main document URL associated with the request.

- `timeoutInterval` (page 1476)
  Returns the receiver's timeout interval, in seconds.

- `networkServiceType` (page 1475)
  Returns the network service type of the request.

- `URL` (page 1476)
  Returns the request's URL.

Getting HTTP Request Properties

- `allHTTPHeaderFields` (page 1472)
  Returns a dictionary containing all the receiver's HTTP header fields.

- `HTTPBody` (page 1472)
  Returns the receiver's HTTP body data.

- `HTTPBodyStream` (page 1473)
  Returns the receiver's HTTP body stream.

- `HTTPMethod` (page 1473)
  Returns the receiver's HTTP request method.

- `HTTPShouldHandleCookies` (page 1473)
  Returns whether the default cookie handling will be used for this request.

- valueForHTTPHeaderField: (page 1476)
  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

See Also
+ requestWithURL:cachePolicy:timeoutInterval: (page 1471)

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

See Also
- initWithURL:cachePolicy:timeoutInterval: (page 1475)

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

See Also
- valueForHTTPHeaderField: (page 1476)

Declared In
NSURLRequest.h

cachePolicy

Returns the receiver's cache policy.

```
- (NSURLRequestCachePolicy)cachePolicy
```

Return Value
The receiver's cache policy.

Availability

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

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

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

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

Declared In
`NSURLRequest.h`

HTTPShouldUsePipelining

Returns whether the request should continue transmitting data before receiving a response from an earlier transmission.

```
- (BOOL)HTTPShouldUsePipelining
```

Return Value
`YES` if the request should continue transmitting data; otherwise, `NO`.

Availability
Available in iOS 4.0 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

See Also
`- initWithURL:cachePolicy:timeoutInterval:` (page 1475)

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

See Also
`- initWithURL:` (page 1474)

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

Declared In
`NSURLRequest.h`

networkServiceType

Returns the network service type of the request.

```
- (NSURLRequestNetworkServiceType)networkServiceType
```

Return Value
The network service type of the request.

Availability
Available in iOS 4.0 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

Declared In
`NSURLRequest.h`

URL

Returns the request's URL.

```
- (NSURL *)URL
```

Return Value
The request's URL.

Availability

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

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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSURLRequest.h`.

`NSURLRequestReloadIgnoringCacheData`
Replaced by `NSURLRequestReloadIgnoringLocalCacheData` (page 1477).

Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSURLRequest.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSURLRequest.h`

NSURLRequestNetworkServiceType

These constants are used to specify the network service type of a request.

```
enum
{
    NSURLNetworkServiceTypeDefault = 0,
    NSURLNetworkServiceTypeVoIP = 1
};
typedef NSUInteger NSURLRequestNetworkServiceType;
```

Constants
`NSURLNetworkServiceTypeDefault`
Specifies standard network traffic.

Available in iOS 4.0 and later.

Declared in `NSURLRequest.h`.

`NSURLNetworkServiceTypeVoIP`
Specifies that the request is providing VoIP service.

Available in iOS 4.0 and later.

Declared in `NSURLRequest.h`.

Availability
Available in iOS 4.0 and later.

Declared In
`NSURLRequest.h`

NSURLResponse Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLResponse.h
Companion guide	URL Loading System Programming Guide

Overview

`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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `copyWithZone:` (page 1554)

Tasks

Creating a Response

- `initWithURL:MIMEType:expectedContentLength:textEncodingName:` (page 1483)
  Returns an initialized `NSURLResponse` object with the URL, MIME type, length, and text encoding set to given values.

Getting the Response Properties

- `expectedContentLength` (page 1482)
  Returns the receiver's expected content length
- `suggestedFilename` (page 1484)
  Returns a suggested filename for the response data.
- `MIMEType` (page 1483)
  Returns the receiver's MIME type.
- `textEncodingName` (page 1484)
  Returns the name of the receiver's text encoding provided by the response's originating source.
- `URL` (page 1484)
  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

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

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

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

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

Declared In
`NSURLResponse.h`

URL

Returns the receiver’s URL.

```
- (NSURL *)URL
```

Return Value
The receiver's URL.

Availability

Declared In
`NSURLResponse.h`

Constants

Response Length Unknown Error

The following error code is returned by `expectedContentLength` (page 1482).

```
#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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSUserDefaults.h
Companion guide	User Defaults Programming Topics
Related sample code	AddMusic MoviePlayer ToolbarSearch

Overview

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 1507) method, which is automatically invoked at periodic intervals, keeps the in-memory cache in sync with a user's defaults database.

The `NSUserDefaults` class provides convenience methods for accessing common types such as floats, doubles, integers, Booleans, and URLs. A default object 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*.

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 1506) 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.

Persistence of NSURL and file reference URLs

When using NSURL instances to refer to files within a process, it's important to make the distinction between location-based tracking (file: scheme URLs that are basically paths) versus filesystem identity tracking (file: scheme URLs that are file reference URLs). When persisting an NSURL, you should take that behavior into consideration. If your application tracks the resource being located by its identity so that it can be found if the user moves the file, then you should explicitly write the NSURL's bookmark data or encode a file reference URL.

If you want to track a file by reference but you require explicit control over when resolution occurs, you should take care to write out bookmark data to NSUserDefaults rather than rely on -[NSUserDefaults setURL:forKey:]. This allows you to call +[NSURL URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:] at a time when you know your application will be able to handle the potential I/O or required user interface interactions.

Tasks

Getting the Shared NSUserDefaults Instance

`+ standardUserDefaults` (page 1491)
Returns the shared defaults object.

`+ resetStandardUserDefaults` (page 1491)
Synchronizes any changes made to the shared user defaults object and releases it from memory.

Initializing an NSUserDefaults Object

`- init` (page 1496)
Returns an `NSUserDefaults` object initialized with the defaults for the current user account.

`- initWithUser:` (page 1496)
Returns an `NSUserDefaults` object initialized with the defaults for the specified user account.

Registering Defaults

- `registerDefaults:` (page 1500)
  Adds the contents of the specified dictionary to the registration domain.

Getting Default Values

- `arrayForKey:` (page 1492)
  Returns the array associated with the specified key.
- `boolForKey:` (page 1493)
  Returns the Boolean value associated with the specified key.
- `dataForKey:` (page 1493)
  Returns the data object associated with the specified key.
- `dictionaryForKey:` (page 1494)
  Returns the dictionary object associated with the specified key.
- `floatForKey:` (page 1495)
  Returns the floating-point value associated with the specified key.
- `integerForKey:` (page 1497)
  Returns the integer value associated with the specified key..
- `objectForKey:` (page 1497)
  Returns the object associated with the first occurrence of the specified default.
- `stringArrayForKey:` (page 1506)
  Returns the array of strings associated with the specified key.
- `stringForKey:` (page 1506)
  Returns the string associated with the specified key.
- `doubleForKey:` (page 1495)
  Returns the double value associated with the specified key.
- `URLForKey:` (page 1507)
  Returns the `NSURL` instance associated with the specified key.

Setting Default Values

- `setBool:forKey:` (page 1502)
  Sets the value of the specified default key to the specified Boolean value.
- `setFloat:forKey:` (page 1502)
  Sets the value of the specified default key to the specified floating-point value.
- `setInteger:forKey:` (page 1503)
  Sets the value of the specified default key to the specified integer value.
- `setObject:forKey:` (page 1503)
  Sets the value of the specified default key in the standard application domain.
- `setDouble:forKey:` (page 1502)
  Sets the value of the specified default key to the double value.

- `setURL:forKey:` (page 1504)
  Sets the value of the specified default key to the specified URL.

Removing Defaults

- `removeObjectForKey:` (page 1500)
  Removes the value of the specified default key in the standard application domain.

Maintaining Persistent Domains

- `synchronize` (page 1507)
  Writes any modifications to the persistent domains to disk and updates all unmodified persistent domains to what is on disk.
- `persistentDomainForName:` (page 1499)
  Returns a dictionary containing the keys and values in the specified persistent domain.
- `persistentDomainNames` (page 1499)
  Returns an array of the current persistent domain names.
- `removePersistentDomainForName:` (page 1500)
  Removes the contents of the specified persistent domain from the user's defaults.
- `setPersistentDomain:forName:` (page 1504)
  Sets the dictionary for the specified persistent domain.

Accessing Managed Environment Keys

- `objectIsForcedForKey:` (page 1498)
  Returns a Boolean value indicating whether the specified key is managed by an administrator.
- `objectIsForcedForKey:inDomain:` (page 1498)
  Returns a Boolean value indicating whether the key in the specified domain is managed by an administrator.

Managing the Search List

- `dictionaryRepresentation` (page 1494)
  Returns a dictionary that contains a union of all key-value pairs in the domains in the search list.

Maintaining Volatile Domains

- `removeVolatileDomainForName:` (page 1501)
  Removes the specified volatile domain from the user's defaults.
- `setVolatileDomain:forName:` (page 1505)
  Sets the dictionary for the specified volatile domain.
- `volatileDomainForName:` (page 1508)
  Returns the dictionary for the specified volatile domain.

- `volatileDomainNames` (page 1508)
  Returns an array of the current volatile domain names.

Maintaining Suites

- `addSuiteNamed:` (page 1492)
  Inserts the specified domain name into the receiver's search list.
- `removeSuiteNamed:` (page 1501)
  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 1491) creates a new shared user defaults object with the standard search list.

Availability
Available in iOS 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 1496) or `init` (page 1496).

Availability
Available in iOS 2.0 and later.

Related Sample Code
MoviePlayer
ToolbarSearch

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 iOS 2.0 and later.

See Also
`+ standardUserDefaults` (page 1491)
`- removeSuiteNamed:` (page 1501)

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 iOS 2.0 and later.

See Also
`- setObject:forKey:` (page 1503)

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
If a boolean value is associated with *defaultName* in the user defaults, that value is returned. Otherwise, `NO` is returned.

Availability
Available in iOS 2.0 and later.

See Also
`- setBool:forKey:` (page 1502)

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 iOS 2.0 and later.

See Also

- `setObject:forKey:` (page 1503)

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 iOS 2.0 and later.

See Also

- `setObject:forKey:` (page 1503)

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 1497), 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 iOS 2.0 and later.

Declared In
NSUserDefaults.h

doubleForKey:

Returns the double value associated with the specified key.

```
- (double)doubleForKey:(NSString *)defaultName
```

Parameters

defaultName
A key in the current user's defaults database.

Return Value
The double value associated with the specified key. If the key does not exist, this method returns `0`.

Availability
Available in iOS 2.0 and later.

See Also
`- setDouble:forKey:` (page 1502)

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 key does not exist, this method returns `0`.

Availability
Available in iOS 2.0 and later.

See Also
`- setFloat:forKey:` (page 1502)

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 iOS 2.0 and later.

See Also
`+ standardUserDefaults` (page 1491)

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 iOS 2.0 and later.

See Also
`+ standardUserDefaults` (page 1491)

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 specified key does not exist, this method returns `0`.

Availability

Available in iOS 2.0 and later.

See Also

`- setInteger:forKey:` (page 1503)

Related Sample Code

MoviePlayer

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 iOS 2.0 and later.

See Also

`- arrayForKey:` (page 1492)
`- dataForKey:` (page 1493)
`- dictionaryForKey:` (page 1494)
`- stringArrayForKey:` (page 1506)
`- stringForKey:` (page 1506)

Related Sample Code
ToolbarSearch

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 iOS 2.0 and later.

See Also
`- objectIsForcedForKey:inDomain:` (page 1498)

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 iOS 2.0 and later.

See Also
- `objectIsForcedForKey:` (page 1498)

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 iOS 2.0 and later.

See Also
- `removePersistentDomainForName:` (page 1500)
- `setPersistentDomain:forName:` (page 1504)

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 1499) method.

Availability
Available in iOS 2.0 and later.

See Also
- `removePersistentDomainForName:` (page 1500)
- `setPersistentDomain:forName:` (page 1504)

Declared In
`NSUserDefaults.h`

registerDefaults:

Adds the contents of 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 iOS 2.0 and later.

Related Sample Code

MoviePlayer

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 1497) method if the same key exists in a domain that precedes the standard application domain in the search list.

Availability

Available in iOS 2.0 and later.

See Also

`- setObject:forKey:` (page 1503)

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 1510) is posted.

Availability

Available in iOS 2.0 and later.

See Also

`- setPersistentDomain:forName:` (page 1504)

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 iOS 2.0 and later.

See Also

`- addSuiteNamed:` (page 1492)

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 iOS 2.0 and later.

See Also

`- setVolatileDomain:forName:` (page 1505)

Declared In
NSUserDefaults.h

setBool:forKey:

Sets the value of the specified default key to the specified Boolean value.

```
- (void)setBool:(BOOL)value forKey:(NSString *)defaultName
```

Parameters

value
The Boolean value to store in the defaults database.

defaultName
The key with which to associate with the value.

Discussion
Invokes `setObject:forKey:` (page 1503) as part of its implementation.

Availability
Available in iOS 2.0 and later.

See Also
`- boolForKey:` (page 1493)

Declared In
NSUserDefaults.h

setDouble:forKey:

Sets the value of the specified default key to the double value.

```
- (void)setDouble:(double)value forKey:(NSString *)defaultName
```

Parameters

value
The double value.

defaultName
The key with which to associate with the value.

Availability
Available in iOS 2.0 and later.

Declared In
NSUserDefaults.h

setFloat:forKey:

Sets the value of the specified default key to the specified floating-point value.

```
- (void)setFloat:(float)value forKey:(NSString *)defaultName
```

Parameters

value

The floating-point value to store in the defaults database.

defaultName

The key with which to associate with the value.

Discussion

Invokes `setObject:forKey:` (page 1503) as part of its implementation.

Availability

Available in iOS 2.0 and later.

See Also

`- floatForKey:` (page 1495)

Declared In

`NSUserDefaults.h`

setInteger:forKey:

Sets the value of the specified default key to the specified integer value.

```
- (void)setInteger:(NSInteger)value forKey:(NSString *)defaultName
```

Parameters

value

The integer value to store in the defaults database.

defaultName

The key with which to associate with the value.

Discussion

Invokes `setObject:forKey:` (page 1503) as part of its implementation.

Availability

Available in iOS 2.0 and later.

See Also

`- integerForKey:` (page 1497)

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.

defaultName
The key with which to associate with the value.

Discussion
The `value` parameter can be only property list objects: `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, or `NSDictionary`. For `NSArray` and `NSDictionary` objects, their contents must be property list objects. See "What is a Property List?" in *Property List Programming Guide*.

Setting a default has no effect on the value returned by the `objectForKey:` (page 1497) method if the same key exists in a domain that precedes the application domain in the search list.

Availability
Available in iOS 2.0 and later.

See Also
`- removeObjectForKey:` (page 1500)

Related Sample Code
ToolbarSearch

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 1510) is posted.

Availability
Available in iOS 2.0 and later.

See Also
`- persistentDomainForName:` (page 1499)
`- persistentDomainNames` (page 1499)

Declared In
`NSUserDefaults.h`

setURL:forKey:

Sets the value of the specified default key to the specified URL.

```
- (void)setURL:(NSURL *)url forKey:(NSString *)defaultName
```

Parameters

url

The `NSURL` to store in the defaults database.

defaultName

The key with which to associate with the value.

Discussion

When an NSURL is stored using -[NSUserDefaults setURL:forKey:], some adjustments are made:

1. Any non-file URL is written by calling `+[NSKeyedArchiver archivedDataWithRootObject:]` using the NSURL instance as the root object.
2. Any file reference `file:` scheme URL will be treated as a non-file URL, and information which makes this URL compatible with 10.5 systems will also be written as part of the archive as well as its minimal bookmark data.
3. Any path-based file: scheme URL is written by first taking the absolute URL, getting the path from that and then determining if the path can be made relative to the user's home directory. If it can, the string is abbreviated by using `stringByAbbreviatingWithTildeInPath` (page 1262) and written out. This allows pre-10.6 clients to read the default and use `-[NSString stringByExpandingTildeInPath]` to use this information.

Availability

Available in iOS 4.0 and later.

See Also

`- URLForKey:` (page 1507)

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 iOS 2.0 and later.

See Also
- `volatileDomainForName:` (page 1508)
- `volatileDomainNames` (page 1508)

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 iOS 2.0 and later.

See Also
- `setObject:forKey:` (page 1503)

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 iOS 2.0 and later.

See Also
- `setObject:forKey:` (page 1503)

Related Sample Code
MoviePlayer

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 iOS 2.0 and later.

See Also
- `persistentDomainForName:` (page 1499)
- `persistentDomainNames` (page 1499)
- `removePersistentDomainForName:` (page 1500)
- `setPersistentDomain:forName:` (page 1504)

Related Sample Code
MoviePlayer

Declared In
`NSUserDefaults.h`

URLForKey:

Returns the `NSURL` instance associated with the specified key.

```
- (NSURL *)URLForKey:(NSString *)defaultName
```

Parameters
defaultName
A key in the current user's defaults database.

Return Value
The `NSURL` instance value associated with the specified key. If the key does not exist, this method returns `nil`.

Discussion
When an NSURL is read using -[NSUserDefaults URLForKey:], the following logic is used:

1. If the value for the key is an `NSData`, the `NSData` is used as the argument to `+[NSKeyedUnarchiver unarchiveObjectWithData:]`. If the `NSData` can be unarchived as an NSURL, the NSURL is returned otherwise nil is returned.
2. If the value for this key was a file reference URL, the file reference URL will be created but its bookmark data will not be resolved until the NSURL instance is later used (e.g. at `-[NSData initWithContentsOfURL:]`).
3. If the value for the key is an NSString which begins with a ~, the NSString will be expanded using `-[NSString stringByExpandingTildeInPath]` and a file: scheme NSURL will be created from that.

Availability
Available in iOS 4.0 and later.

See Also
`- setURL:forKey:` (page 1504)

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 iOS 2.0 and later.

See Also
`- removeVolatileDomainForName:` (page 1501)
`- setVolatileDomain:forName:` (page 1505)

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 1508) method.

Availability
Available in iOS 2.0 and later.

See Also
- `removeVolatileDomainForName:` (page 1501)
- `setVolatileDomain:forName:` (page 1505)

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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSValue.h Foundation/NSGeometry.h Foundation/NSRange.h
Companion guide	Number and Value Programming Topics
Related sample code	KeyboardAccessory

Overview

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 1552)
- `initWithCoder:` (page 1552)

NSCopying
- `- copyWithZone:` (page 1554)

Tasks

Creating an NSValue

- `initWithBytes:objCType:` (page 1516)
  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 1513)
  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 1513)
  Creates and returns an `NSValue` object that contains a given value which is interpreted as being of a given Objective-C type.
+ `valueWithNonretainedObject:` (page 1514)
  Creates and returns an `NSValue` object that contains a given object.
+ `valueWithPointer:` (page 1514)
  Creates and returns an `NSValue` object that contains a given pointer.
+ `valueWithRange:` (page 1515)
  Creates and returns an `NSValue` object that contains a given `NSRange` structure.

Accessing Data

- `getValue:` (page 1515)
  Copies the receiver's value into a given buffer.
- `nonretainedObjectValue` (page 1517)
  Returns the receiver's value as an `id`.
- `objCType` (page 1517)
  Returns a C string containing the Objective-C type of the data contained in the receiver.
- `pointerValue` (page 1517)
  Returns the receiver's value as a pointer to `void`.
- `rangeValue` (page 1518)
  Returns an `NSRange` structure representation of the receiver.

Comparing Objects

- `isEqualToValue:` (page 1516)
  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 1513) and may be deprecated in a future release. You should use `valueWithBytes:objCType:` (page 1513) instead.

Availability

Available in iOS 2.0 and later.

See Also

`+ valueWithBytes:objCType:` (page 1513)

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 other considerations in creating an `NSValue` object and code examples.

Availability
Available in iOS 2.0 and later.

See Also
- `initWithBytes:objCType:` (page 1516)

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 1513) 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 iOS 2.0 and later.

See Also
- `nonretainedObjectValue` (page 1517)

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 1513) 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 iOS 2.0 and later.

See Also
- pointerValue (page 1517)

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 iOS 2.0 and later.

See Also
- rangeValue (page 1518)

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 iOS 2.0 and later.

Related Sample Code
KeyboardAccessory

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 other considerations in creating an `NSValue` object.

This is the designated initializer for the `NSValue` class.

Availability
Available in iOS 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 iOS 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 iOS 2.0 and later.

See Also
- getValue: (page 1515)

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 iOS 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 iOS 2.0 and later.

See Also
- getValue: (page 1515)

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 iOS 2.0 and later.

See Also
+ valueWithRange: (page 1515)

Declared In
NSRange.h

NSValueTransformer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Companion guide	Value Transformer Programming Guide
Availability	Available in iOS 3.0 and later.

Overview

`NSValueTransformer` is an abstract class that is used by the Cocoa Bindings technology to transform values from one representation to another.

An application creates a subclass of `NSValueTransformer`, overriding the necessary methods to provide the required custom transformation.

Example

A relatively trivial value transformer takes an object of type `id` and returns a string based on the object'□ □ s class type. This transformer is not reversible as it's probably unreasonable to transform a class name into an object. The value transformer class you write to accomplish this simple task could look like:

```
@interface ClassNameTransformer: NSValueTransformer {}
@end
@implementation ClassNameTransformer
+ (Class)transformedValueClass { return [NSString class]; }
+ (BOOL)allowsReverseTransformation { return NO; }
- (id)transformedValue:(id)value {
    return (value == nil) ? nil : NSStringFromClass([value class]);
}
@end
```

Tasks

Using Name-based Registry

`+ setValueTransformer:forName:` (page 1521)
Registers the value transformer a given transformer with a given identifier.

`+ valueTransformerForName:` (page 1521)
Returns the value transformer identified by a given identifier.

`+ valueTransformerNames` (page 1522)
Returns an array of all the registered value transformers.

Getting Information About a Transformer

`+ allowsReverseTransformation` (page 1520)
Returns a Boolean value that indicates whether the receiver can reverse a transformation.

`+ transformedValueClass` (page 1521)
Returns the class of the value returned by the receiver for a forward transformation.

Using Transformers

`- transformedValue:` (page 1523)
Returns the result of transforming a given value.

`- reverseTransformedValue:` (page 1522)
Returns the result of the reverse transformation of a given value.

Class Methods

allowsReverseTransformation

Returns a Boolean value that indicates whether the receiver can reverse a transformation.

```
+ (BOOL)allowsReverseTransformation
```

Return Value
`YES` if the receiver supports reverse value transformations, otherwise `NO`.

The default is `NO`.

Discussion
A subclass should override this method to return `YES` if it supports reverse value transformations.

Availability
Available in iOS 3.0 and later.

Declared In
NSValueTransformer.h

setValueTransformer:forName:

Registers the value transformer a given transformer with a given identifier.

```
+ (void)setValueTransformer:(NSValueTransformer *)transformer forName:(NSString
    *)name
```

Parameters

transformer
The transformer to register.

name
The name for *transformer*.

Availability
Available in iOS 3.0 and later.

See Also
+ valueTransformerForName: (page 1521)

Declared In
NSValueTransformer.h

transformedValueClass

Returns the class of the value returned by the receiver for a forward transformation.

```
+ (Class)transformedValueClass
```

Return Value
The class of the value returned by the receiver for a forward transformation.

Discussion
A subclass should override this method to return the appropriate class.

Availability
Available in iOS 3.0 and later.

Declared In
NSValueTransformer.h

valueTransformerForName:

Returns the value transformer identified by a given identifier.

```
+ (NSValueTransformer *)valueTransformerForName:(NSString *)name
```

Parameters

name
The transformer identifier.

Return Value
The value transformer identified by *name* in the shared registry, or `nil` if not found.

Discussion
If `valueTransformerForName:` does not find a registered transformer instance for *name*, it will attempt to find a class with the specified name. If a corresponding class is found an instance will be created and initialized using its `init:` method and then automatically registered with `name`.

Availability
Available in iOS 3.0 and later.

See Also
`+ setValueTransformer:forName:` (page 1521)

Declared In
`NSValueTransformer.h`

valueTransformerNames

Returns an array of all the registered value transformers.

```
+ (NSArray *)valueTransformerNames
```

Return Value
An array of all the registered value transformers.

Availability
Available in iOS 3.0 and later.

Declared In
`NSValueTransformer.h`

Instance Methods

reverseTransformedValue:

Returns the result of the reverse transformation of a given value.

```
- (id)reverseTransformedValue:(id)value
```

Parameters

value

The value to reverse transform.

Return Value
The reverse transformation of *value*.

Discussion
The default implementation raises an exception if `allowsReverseTransformation` (page 1520) returns `NO`; otherwise it will invoke `transformedValue:` (page 1523) with *value*.

A subclass should override this method if they require a reverse transformation that is not the same as simply reapplying the original transform (as would be the case with negation, for example). For example, if a value transformer converts a value in Fahrenheit to Celsius, this method would converts a value from Celsius to Fahrenheit.

Availability
Available in iOS 3.0 and later.

See Also
`- transformedValue:` (page 1523)

Declared In
`NSValueTransformer.h`

transformedValue:

Returns the result of transforming a given value.

```
- (id)transformedValue:(id)value
```

Parameters

value

The value to transform.

Return Value
The result of transforming *value*.

The default implementation simply returns *value*.

Discussion
A subclass should override this method to transform and return an object based on *value*.

Availability
Available in iOS 3.0 and later.

See Also
`- reverseTransformedValue:` (page 1522)

Declared In
`NSValueTransformer.h`

Constants

Named Value Transformers

The following named value transformers are defined by `NSValueTransformer`:

```
NSString * const NSNegateBooleanTransformerName;
NSString * const NSIsNilTransformerName ;
NSString * const NSIsNotNilTransformerName ;
NSString * const NSUnarchiveFromDataTransformerName ;
NSString * const NSKeyedUnarchiveFromDataTransformerName ;
```

Constants

`NSNegateBooleanTransformerName`

This value transformer negates a boolean value, transforming `YES` to `NO` and `NO` to `YES`.

This transformer is reversible.

Available in iOS 3.0 and later.

Declared in `NSValueTransformer.h`.

`NSIsNilTransformerName`

This value transformer returns `YES` if the value is `nil`.

This transformer is not reversible.

Available in iOS 3.0 and later.

Declared in `NSValueTransformer.h`.

`NSIsNotNilTransformerName`

This value transformer returns `YES` if the value is non-`nil`.

This transformer is not reversible.

Available in iOS 3.0 and later.

Declared in `NSValueTransformer.h`.

`NSUnarchiveFromDataTransformerName`

This value transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value.

The reverse transformation returns an `NSData` instance created by archiving the value. The archived object must implement the `NSCoding` protocol using sequential archiving in order to be unarchived and archived with this transformer.

Available in iOS 3.0 and later.

Declared in `NSValueTransformer.h`.

`NSKeyedUnarchiveFromDataTransformerName`

This value transformer returns an object created by attempting to unarchive the data in the `NSData` object passed as the value. The archived object must be created using keyed archiving in order to be unarchived and archived with this transformer.

The reverse transformation returns an `NSData` instance created by archiving the value using keyed archiving. The archived object must implement the `NSCoding` protocol using keyed archiving in order to be unarchived and archived with this transformer.

Available in iOS 3.0 and later.

Declared in `NSValueTransformer.h`.

Declared In

`NSValueTransformer.h`

NSXMLParser Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSXMLParser.h
Companion guide	Event-Driven XML Programming Guide
Related sample code	GKTank

Overview

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 1528)
  Initializes the receiver with the XML content referenced by the given URL.
- `initWithData:` (page 1528)
  Initializes the receiver with the XML contents encapsulated in a given data object.

Managing Delegates

- `setDelegate:` (page 1530)
  Sets the receiver's delegate.
- `delegate` (page 1527)
  Returns the receiver's delegate.

Managing Parser Behavior

- `setShouldProcessNamespaces:` (page 1531)
  Specifies whether the receiver reports the namespace and the qualified name of an element in related delegation methods .
- `shouldProcessNamespaces` (page 1532)
  Indicates whether the receiver reports the namespace and the qualified name of an element in related delegation methods.
- `setShouldReportNamespacePrefixes:` (page 1531)
  Specifies whether the receiver reports the scope of namespace declarations using related delegation methods.
- `shouldReportNamespacePrefixes` (page 1533)
  Indicates whether the receiver reports the prefixes indicating the scope of namespace declarations using related delegation methods.
- `setShouldResolveExternalEntities:` (page 1532)
  Specifies whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1661).
- `shouldResolveExternalEntities` (page 1533)
  Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1661).

Parsing

- `parse` (page 1529)
  Starts the event-driven parsing operation.
- `abortParsing` (page 1527)
  Stops the parser object.
- `parserError` (page 1529)
  Returns an `NSError` object from which you can obtain information about a parsing error.

Obtaining Parser State

- `columnNumber` (page 1527)
  Returns the column number of the XML document being processed by the receiver.
- `lineNumber` (page 1529)
  Returns the line number of the XML document being processed by the receiver.

- `publicID` (page 1530)
  Returns the public identifier of the external entity referenced in the XML document.
- `systemID` (page 1533)
  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 1664), is informed of the cancelled parsing operation.

Availability
Available in iOS 2.0 and later.

See Also
- `parse` (page 1529)
- `parserError` (page 1529)

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 iOS 2.0 and later.

See Also
- `lineNumber` (page 1529)

Declared In
`NSXMLParser.h`

delegate

Returns the receiver's delegate.

```
- (id < NSXMLParserDelegate >)delegate
```

Availability
Available in iOS 2.0 and later.

See Also
- setDelegate: (page 1530)

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 iOS 2.0 and later.

See Also
- initWithData: (page 1528)

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 iOS 2.0 and later.

See Also
- `initWithContentsOfURL:` (page 1528)

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 iOS 2.0 and later.

See Also
- `columnNumber` (page 1527)

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 iOS 2.0 and later.

See Also
- `abortParsing` (page 1527)
- `parserError` (page 1529)

Related Sample Code
GKTank

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 iOS 2.0 and later.

See Also
- `abortParsing` (page 1527)
- `parse` (page 1529)

Related Sample Code
GKTank

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 iOS 2.0 and later.

See Also
- `systemID` (page 1533)

Declared In
`NSXMLParser.h`

setDelegate:

Sets the receiver's delegate.

```
- (void)setDelegate:(id < NSXMLParserDelegate >)delegate
```

Parameters
delegate
An object that is the new delegate. It is not retained. The delegate must conform to the `NSXMLParserDelegate Protocol` protocol.

Availability
Available in iOS 2.0 and later.

See Also
- `delegate` (page 1527)

Related Sample Code
GKTank

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 1657) and parser:didEndElement:namespaceURI:qualifiedName: (page 1656).

Availability
Available in iOS 2.0 and later.

See Also
- shouldProcessNamespaces (page 1532)

Related Sample Code
GKTank

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 1658) and parser:didEndMappingPrefix: (page 1657).

Availability
Available in iOS 2.0 and later.

See Also
- shouldReportNamespacePrefixes (page 1533)

Related Sample Code
GKTank

Declared In
NSXMLParser.h

setShouldResolveExternalEntities:

Specifies whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1661).

```
- (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 iOS 2.0 and later.

See Also
`- shouldResolveExternalEntities` (page 1533)

Related Sample Code
GKTank

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 1657) and `parser:didEndElement:namespaceURI:qualifiedName:` (page 1656).

Availability
Available in iOS 2.0 and later.

See Also
- `setShouldProcessNamespaces:` (page 1531)

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 1658) and `parser:didEndMappingPrefix:` (page 1657).

Availability
Available in iOS 2.0 and later.

See Also
- `setShouldReportNamespacePrefixes:` (page 1531)

Declared In
`NSXMLParser.h`

shouldResolveExternalEntities

Indicates whether the receiver reports declarations of external entities using the delegate method `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1661).

```
- (BOOL)shouldResolveExternalEntities
```

Return Value
`YES` if the receiver reports declarations of external entities, `NO` otherwise. The default value is `NO`.

Availability
Available in iOS 2.0 and later.

See Also
- `setShouldResolveExternalEntities:` (page 1532)

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 iOS 2.0 and later.

See Also
- publicID (page 1530)

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 iOS 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 1534).

```
typedef NSInteger NSXMLParserError;
```

Availability
Available in iOS 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 iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserOutOfMemoryError`

The parser object ran out of memory.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDocumentStartError`

The parser object is unable to start parsing.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEmptyDocumentError`
The document is empty.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserPrematureDocumentEndError`
The document ended unexpectedly.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidHexCharacterRefError`
Invalid hexadecimal character reference encountered.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidDecimalCharacterRefError`
Invalid decimal character reference encountered.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterRefError`
Invalid character reference encountered.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterError`
Invalid character encountered.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefAtEOFError`
Target of character reference cannot be found.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInPrologError`
Invalid character found in the prolog.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInEpilogError`
Invalid character found in the epilog.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCharacterRefInDTDError`
Invalid character encountered in the DTD.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefAtEOFError`
Target of entity reference is not found.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInPrologError`
Invalid entity reference found in the prolog.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInEpilogError`
Invalid entity reference found in the epilog.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefInDTDError`
Invalid entity reference found in the DTD.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefAtEOFError`
Target of parsed entity reference is not found.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInPrologError`
Target of parsed entity reference is not found in prolog.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInEpilogError`
Target of parsed entity reference is not found in epilog.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInInternalSubsetError`
Target of parsed entity reference is not found in internal subset.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityReferenceWithoutNameError`
Entity reference is without name.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityReferenceMissingSemiError`
Entity reference is missing semicolon.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefNoNameError`
Parsed entity reference is without an entity name.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefMissingSemiError`
Parsed entity reference is missing semicolon.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserUndeclaredEntityError`
Entity is not declared.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserUnparsedEntityError`
Cannot parse entity.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityIsExternalError`
Cannot parse external entity.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityIsParameterError`
Entity is a parameter.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserUnknownEncodingError`
Document encoding is unknown.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEncodingNotSupportedError`
Document encoding is not supported.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserStringNotStartedError`
String is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserStringNotClosedError`
String is not closed.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNamespaceDeclarationError`
Invalid namespace declaration encountered.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityNotStartedError`
Entity is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityNotFinishedError`
Entity is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLessThanSymbolInAttributeError`
Angle bracket is used in attribute.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeNotStartedError`
Attribute is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeNotFinishedError`
Attribute is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeHasNoValueError`
Attribute doesn't contain a value.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeRedefinedError`
Attribute is redefined.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLiteralNotStartedError`
Literal is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLiteralNotFinishedError`
Literal is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCommentNotFinishedError`
Comment is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserProcessingInstructionNotStartedError`
Processing instruction is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserProcessingInstructionNotFinishedError`
Processing instruction is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotationNotStartedError`
Notation is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotationNotFinishedError`
Notation is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeListNotStartedError`
Attribute list is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserAttributeListNotFinishedError`
Attribute list is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserMixedContentDeclNotStartedError`
Mixed content declaration is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserMixedContentDeclNotFinishedError`
Mixed content declaration is not finished.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserElementContentDeclNotStartedError`
Element content declaration is not started.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserElementContentDeclNotFinishedError`
Element content declaration is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserXMLDeclNotStartedError`
XML declaration is not started.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserXMLDeclNotFinishedError`
XML declaration is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserConditionalSectionNotStartedError`
Conditional section is not started.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserConditionalSectionNotFinishedError`
Conditional section is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserExternalSubsetNotFinishedError`
External subset is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDOCTYPEDeclNotFinishedError`
Document type declaration is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserMisplacedCDATAEndStringError`
Misplaced CDATA end string.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserCDATANotFinishedError`
CDATA block is not finished.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserMisplacedXMLDeclarationError`
Misplaced XML declaration.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserSpaceRequiredError`
Space is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserSeparatorRequiredError`
Separator is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNMTOKENRequiredError`
Name token is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNAMERequiredError`
Name is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserPCDATARequiredError`
CDATA is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserURIRequiredError`
URI is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserPublicIdentifierRequiredError`
Public identifier is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLTRequiredError`
Left angle bracket is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserGTRequiredError`
Right angle bracket is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserLTSlashRequiredError`
Left angle bracket slash is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEqualExpectedError`
Equal sign expected.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserTagNameMismatchError`
Tag name mismatch.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserUnfinishedTagError`
Unfinished tag found.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserStandaloneValueError`
Standalone value found.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidEncodingNameError`
Invalid encoding name found.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserCommentContainsDoubleHyphenError`
Comment contains double hyphen.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidEncodingError`
Invalid encoding.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserExternalStandaloneEntityError`
External standalone entity.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserInvalidConditionalSectionError`
Invalid conditional section.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserEntityValueRequiredError`
Entity value is required.
Available in iOS 2.0 and later.
Declared in `NSXMLParser.h`.

`NSXMLParserNotWellBalancedError`
Document is not well balanced.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserExtraContentError`
Error in content found.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidCharacterInEntityError`
Invalid character in entity found.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserParsedEntityRefInInternalError`
Internal error in parsed entity reference found.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityRefLoopError`
Entity reference loop encountered.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserEntityBoundaryError`
Entity boundary error.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserInvalidURIError`
Invalid URI specified.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserURIFragmentError`
URI fragment.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserNoDTDError`
Missing DTD.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

`NSXMLParserDelegateAbortedParseError`
Delegate aborted parse.

Available in iOS 2.0 and later.

Declared in `NSXMLParser.h`.

Declared In
`NSXMLParser.h`

Protocols

NSCacheDelegate Protocol Reference

Adopted by	NSCache
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	NSCache.h

Overview

The delegate of an `NSCache` object implements this protocol to perform specialized actions when an object is about to be evicted or removed from the cache.

Tasks

Responding to Object Eviction

`- cache:willEvictObject:` (page 1549)
Called when an object is about to be evicted or removed from the cache.

Instance Methods

cache:willEvictObject:

Called when an object is about to be evicted or removed from the cache.

```
- (void)cache:(NSCache *)cache willEvictObject:(id)obj
```

Parameters

cache
The cache with which the object of interest is associated.

obj
The object of interest in the cache.

Discussion
It is not possible to modify `cache` from within the implementation of this delegate method.

Availability
Available in iOS 4.0 and later.

Declared In
`NSCache.h`

NSCoding Protocol Reference

Adopted by	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Archives and Serializations Programming Guide

Overview

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 1552) or `initWithCoder:` (page 1552). `encodeWithCoder:` (page 1552) instructs the object to encode its instance variables to the coder provided; an object can receive this method any number of times. `initWithCoder:` (page 1552) 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*.

Tasks

Initializing with a Coder

- `initWithCoder:` (page 1552) *required method*

  Returns an object initialized from data in a given unarchiver. (required)

Encoding with a Coder

Instance Methods

encodeWithCoder:

Encodes the receiver using a given archiver. (required)

```
- (void)encodeWithCoder:(NSCoder *)encoder
```

Parameters

encoder

An archiver object.

Availability
Available in iOS 2.0 and later.

Declared In
NSObject.h

initWithCoder:

Returns an object initialized from data in a given unarchiver. (required)

```
- (id)initWithCoder:(NSCoder *)decoder
```

Parameters

decoder

An unarchiver object.

Return Value
self, initialized using the data in *decoder*.

Availability
Available in iOS 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 iOS 2.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Memory Management Programming Guide

Overview

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 1554), 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 1554) with the default zone.

Your options for implementing this protocol are as follows:

- Implement `NSCopying` using `alloc` (page 949) and `init...` in classes that don't inherit `copyWithZone:` (page 1554).
- Implement `NSCopying` by invoking the superclass's `copyWithZone:` (page 1554) when `NSCopying` behavior is inherited. If the superclass implementation might use the `NSCopyObject` (page 1692) 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 1554) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

- `copyWithZone:` (page 1554) *required method*
  Returns a new instance that's a copy of the receiver. (required)

Instance Methods

copyWithZone:

Returns a new instance that's a copy of the receiver. (required)

```
- (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 iOS 2.0 and later.

See Also

- `mutableCopyWithZone:` (page 1614) (NSMutableCopying protocol)
- `copy` (page 965) (NSObject class)

Declared In

`NSObject.h`

NSDecimalNumberBehaviors Protocol Reference

Adopted by	NSDecimalNumberHandler
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSDecimalNumber.h
Companion guide	Number and Value Programming Topics

Overview

The `NSDecimalBehaviors` protocol declares three methods that control the discretionary aspects of working with `NSDecimalNumber` objects.

The `scale` (page 1557) and `roundingMode` (page 1556) 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 1556) 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 1556) *required method*
  Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values. (required)
- `scale` (page 1557) *required method*
  Returns the number of digits allowed after the decimal separator. (required)

Handling Errors

- `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1556) *required method*
  Specifies what an `NSDecimalNumber` object will do when it encounters an error. (required)

Instance Methods

exceptionDuringOperation:error:leftOperand:rightOperand:

Specifies what an `NSDecimalNumber` object will do when it encounters an error. (required)

```
- (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 1559). 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*.
- 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 iOS 2.0 and later.

Declared In

`NSDecimalNumber.h`

roundingMode

Returns the way that `NSDecimalNumber`'s `decimalNumberBy...` methods round their return values. (required)

```
- (NSRoundingMode)roundingMode
```

Return Value
Returns the current rounding mode. See "NSRoundingMode" (page 1557) for possible values.

Availability
Available in iOS 2.0 and later.

Declared In
`NSDecimalNumber.h`

scale

Returns the number of digits allowed after the decimal separator. (required)

```
- (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 1556) 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 iOS 2.0 and later.

Declared In
`NSDecimalNumber.h`

Constants

NSRoundingMode

These constants specify rounding behaviors.

```
enum {
    NSRoundPlain,
    NSRoundDown,
    NSRoundUp,
    NSRoundBankers
};
typedef NSUInteger 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 iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSRoundDown`

Round return values down.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSRoundUp`

Round return values up.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSDecimal.h`.

Discussion

The rounding mode matters only if the `scale` (page 1557) method sets a limit on the precision of `NSDecimalNumber` return values. It has no effect if `scale` returns `NSDecimalNoScale`. Assuming that `scale` (page 1557) 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

NSCalculationError

Calculation error constants used to describe an error in `exceptionDuringOperation:error:leftOperand:rightOperand:` (page 1556).

```
enum {
    NSCalculationNoError = 0,
    NSCalculationLossOfPrecision,
    NSCalculationUnderflow,
    NSCalculationOverflow,
    NSCalculationDivideByZero
};
typedef NSUInteger NSCalculationError;
```

Constants

`NSCalculationNoError`
No error occurred.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationLossOfPrecision`
The number can't be represented in 38 significant digits.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationOverflow`
The number is too large to represent.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationUnderflow`
The number is too small to represent.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSCalculationDivideByZero`
The caller tried to divide by `0`.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

NSDiscardableContent Protocol Reference

Adopted by	NSPurgeableData
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	NSObject.h

Overview

You implement this protocol when a class's objects have subcomponents that can be discarded when not being used, thereby giving an application a smaller memory footprint.

An `NSDiscardableContent` object's life cycle is dependent upon a "counter" variable. An `NSDiscardableContent` object is a purgeable block of memory that keeps track of whether or not it is currently being used by some other object. When this memory is being read, or is still needed, its counter variable will be greater than or equal to 1. When it is not being used, and can be discarded, the counter variable will be equal to 0.

When the counter is equal to 0, the block of memory may be discarded if memory is tight at that point in time. In order to discard the content, call `discardContentIfPossible` (page 1563) on the object, which will free the associated memory if the counter variable equals 0.

By default, `NSDiscardableContent` objects are initialized with their counter equal to 1 to ensure that they are not immediately discarded by the memory-management system. From this point, you must keep track of the counter variable's state. Calling the `beginContentAccess` (page 1562) method increments the counter variable by 1, thus ensuring that the object will not be discarded. When you no longer need the object, decrement its counter by calling `endContentAccess` (page 1563).

The Foundation framework includes the `NSPurgeableData` class, which provides a default implementation of this protocol.

Tasks

Accessing Content

- `beginContentAccess` (page 1562) *required method*
  Returns a Boolean value indicating whether the discardable contents are still available and have been successfully accessed. (required)
- `endContentAccess` (page 1563) *required method*
  Called if the discardable contents are no longer being accessed. (required)

Discarding Content

- `discardContentIfPossible` (page 1563) *required method*
  Called to discard the contents of the receiver if the value of the accessed counter is 0. (required)
- `isContentDiscarded` (page 1563) *required method*
  Returns a Boolean value indicating whether the content has been discarded. (required)

Instance Methods

beginContentAccess

Returns a Boolean value indicating whether the discardable contents are still available and have been successfully accessed. (required)

```
- (BOOL)beginContentAccess
```

Return Value
YES if the discardable contents are still available and have now been successfully accessed; otherwise, `NO`.

Discussion
Call this method if the object's memory is needed or is about to be used. This method increments the counter variable, thus protecting the object's memory from possibly being discarded. The implementing class may decide that this method will try to recreate the contents if they have been discarded and return YES if the re-creation was successful. Implementors of this protocol should raise exceptions if the `NSDiscardableContent` objects are used when the `beginContentAccess` method has not been called on them.

Availability
Available in iOS 4.0 and later.

See Also
- `endContentAccess` (page 1563)

Declared In
`NSObject.h`

discardContentIfPossible

Called to discard the contents of the receiver if the value of the accessed counter is 0. (required)

```
- (void)discardContentIfPossible
```

Discussion
This method should only discard the contents of the object if the value of the accessed counter is 0. Otherwise, it should do nothing.

Availability
Available in iOS 4.0 and later.

See Also
`- isContentDiscarded` (page 1563)

Declared In
`NSObject.h`

endContentAccess

Called if the discardable contents are no longer being accessed. (required)

```
- (void)endContentAccess
```

Discussion
This method decrements the counter variable of the object, which will usually bring the value of the counter variable back down to 0, which allows the discardable contents of the object to be thrown away if necessary.

Availability
Available in iOS 4.0 and later.

See Also
`- beginContentAccess` (page 1562)

Declared In
`NSObject.h`

isContentDiscarded

Returns a Boolean value indicating whether the content has been discarded. (required)

```
- (BOOL)isContentDiscarded
```

Return Value
`YES` if the content has been discarded; otherwise, `NO`.

Availability
Available in iOS 4.0 and later.

See Also
`- discardContentIfPossible` (page 1563)

Declared In
`NSObject.h`

NSErrorRecoveryAttempting Protocol Reference

~~(informal protocol)~~

Framework	/System/Library/Frameworks/Foundation.framework
Declared in	Foundation/NSErrorRecoveryAttempting.h

Overview

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 1566) is invoked. If the error is presented in an application-modal dialog, `attemptRecoveryFromError:optionIndex:` (page 1565) is invoked.

Tasks

Attempting Recovery From Errors

`- attemptRecoveryFromError:optionIndex:delegate:didRecoverSelector:contextInfo:` (page 1566)

Implemented to attempt a recovery from an error noted in an document-modal sheet.

`- attemptRecoveryFromError:optionIndex:` (page 1565)

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 iOS 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 iOS 2.0 and later.

Declared In
`NSError.h`

NSFastEnumeration Protocol Reference

Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSEnumerator.h
Companion guide	The Objective-C Programming Language
Related sample code	FastEnumerationSample

Overview

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 424) to return items one at a time. For more details, see Fast Enumeration.

Tasks

Enumeration

`- countByEnumeratingWithState:objects:count:` (page 1569) *required method*

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. (required)

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. (required)

```
- (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 iOS 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 1569).

Availability
Available in iOS 2.0 and later.

Declared In
`NSEnumerator.h`

NSKeyedArchiverDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide

Overview

The `NSKeyedArchiverDelegate` protocol defines the optional methods implemented by delegates of `NSKeyedArchiver` objects.

Tasks

Encoding Data and Objects

- `archiver:didEncodeObject:` (page 1574)
  Informs the delegate that a given object has been encoded.
- `archiverDidFinish:` (page 1575)
  Notifies the delegate that encoding has finished.
- `archiver:willEncodeObject:` (page 1574)
  Informs the delegate that *`object`* is about to be encoded.
- `archiverWillFinish:` (page 1576)
  Notifies the delegate that encoding is about to finish.
- `archiver:willReplaceObject:withObject:` (page 1575)
  Informs the delegate that one given object is being substituted for another given object.

Instance 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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

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 982):.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

Declared In

NSKeyedArchiver.h

NSKeyedUnarchiverDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSKeyedArchiver.h
Companion guide	Archives and Serializations Programming Guide

Overview

The `NSKeyedUnarchiverDelegate` protocol defines the optional methods implemented by delegates of `NSKeyedUnarchiver` objects.

Tasks

Decoding Objects

`- unarchiver:cannotDecodeObjectOfClassName:originalClasses:` (page 1578)
Informs the delegate that the class with a given name is not available during decoding.

`- unarchiver:didDecodeObject:` (page 1578)
Informs the delegate that a given object has been decoded.

`- unarchiver:willReplaceObject:withObject:` (page 1579)
Informs the delegate that one object is being substituted for another.

Finishing Decoding

`- unarchiverDidFinish:` (page 1579)
Notifies the delegate that decoding has finished.

`- unarchiverWillFinish:` (page 1580)
Notifies the delegate that decoding is about to finish.

Instance 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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

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`, the decoded value will be unchanged (that is, the original object will be decoded).

Discussion
This method is called after *object* has been sent `initWithCoder:` (page 1552) and `awakeAfterUsingCoder:` (page 964).

The delegate may use this method to keep track of the decoded objects.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 1578).

The delegate may use this method if it is keeping track of the encoded or decoded objects.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

Declared In
NSKeyedArchiver.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

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 1586), which sets the value for the property identified by the specified key, and `valueForKey:` (page 1589), 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 1589)
  Returns the value for the property identified by a given key.
- `valueForKeyPath:` (page 1590)
  Returns the value for the derived property identified by a given key path.
- `dictionaryWithValuesForKeys:` (page 1583)
  Returns a dictionary containing the property values identified by each of the keys in a given array.
- `valueForUndefinedKey:` (page 1590)
  Invoked by `valueForKey:` (page 1589) when it finds no property corresponding to a given key.
- `mutableArrayValueForKey:` (page 1583)
  Returns a mutable array proxy that provides read-write access to an ordered to-many relationship specified by a given key.
- `mutableArrayValueForKeyPath:` (page 1584)
  Returns a mutable array that provides read-write access to the ordered to-many relationship specified by a given key path.

- `mutableSetValueForKey:` (page 1585)
  Returns a mutable set proxy that provides read-write access to the unordered to-many relationship specified by a given key.
- `mutableSetValueForKeyPath:` (page 1585)
  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 1587)
  Sets the value for the property identified by a given key path to a given value.
- `setValuesForKeysWithDictionary:` (page 1588)
  Sets properties of the receiver with values from a given dictionary, using its keys to identify the properties.
- `setNilValueForKey:` (page 1586)
  Invoked by `setValue:forKey:` (page 1586) when it's given a `nil` value for a scalar value (such as an `int` or `float`).
- `setValue:forKey:` (page 1586)
  Sets the property of the receiver specified by a given key to a given value.
- `setValue:forUndefinedKey:` (page 1587)
  Invoked by `setValue:forKey:` (page 1586) when it finds no property for a given key.

Changing Default Behavior

+ `accessInstanceVariablesDirectly` (page 1582)
  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 1588)
  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 1589)
  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 iOS 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 1589) for each key in *keys* and substitutes `NSNull` values in the dictionary for returned `nil` values.

Availability
Available in iOS 2.0 and later.

See Also
`- setValuesForKeysWithDictionary:` (page 1588)

Declared In
`NSKeyValueCoding.h`

mutableArrayValueForKey:

Returns a mutable array proxy 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 proxy 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 1589), 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.

The search pattern that `mutableArrayValueForKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability

Available in iOS 2.0 and later.

See Also

`- mutableArrayValueForKeyPath:` (page 1584)

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 1583) for additional details.

Availability

Available in iOS 2.0 and later.

See Also

`- mutableArrayValueForKey:` (page 1583)

Declared In

`NSKeyValueCoding.h`

mutableSetValueForKey:

Returns a mutable set proxy 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 proxy 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 1589), 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.

The search pattern that `mutableSetValueForKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability

Available in iOS 2.0 and later.

See Also

`- mutableArrayValueForKeyPath:` (page 1584)

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 1585) for additional details.

Availability

Available in iOS 2.0 and later.

See Also

`- mutableArrayValueForKey:` (page 1583)

Declared In
NSKeyValueCoding.h

setNilValueForKey:

Invoked by `setValue:forKey:` (page 1586) 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 iOS 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 search pattern that `setValue:forKey:` uses is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability
Available in iOS 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 1589), and sends the final object a `setValue:forKey:` message.

Special Considerations

When using this method, and the destination object does not implement an accessor for the value, the default behavior is for that object to retain *value* rather than copy or assign *value*.

Availability

Available in iOS 2.0 and later.

See Also

`- valueForKeyPath:` (page 1590)

Declared In

`NSKeyValueCoding.h`

setValue:forUndefinedKey:

Invoked by `setValue:forKey:` (page 1586) 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 iOS 2.0 and later.

See Also

`- valueForUndefinedKey:` (page 1590)

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 1586) for each key-value pair, substituting `nil` for `NSNull` values in *keyedValues*.

Availability

Available in iOS 2.0 and later.

See Also

`- dictionaryWithValuesForKeys:` (page 1583)

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 iOS 2.0 and later.

See Also
- `validateValue:forKeyPath:error:` (page 1589)

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 1589) and returns the result of a `validateValue:forKey:error:` message to the final object.

Availability
Available in iOS 2.0 and later.

See Also
- `validateValue:forKey:error:` (page 1588)

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 search pattern that `valueForKey:` uses to find the correct value to return is described in Accessor Search Implementation Details in *Key-Value Coding Programming Guide*.

Availability
Available in iOS 2.0 and later.

See Also
`- valueForKeyPath:` (page 1590)

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 1589) and returns the result of a `valueForKey:` message to the final object.

Availability
Available in iOS 2.0 and later.

See Also
`- setValue:forKeyPath:` (page 1587)

Declared In
`NSKeyValueCoding.h`

valueForUndefinedKey:

Invoked by `valueForKey:` (page 1589) 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 iOS 2.0 and later.

See Also
`- setValue:forUndefinedKey:` (page 1587)

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 1591)

Available in iOS 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 1591).

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 iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSCountKeyValueOperator`
The `@count` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfArraysKeyValueOperator`
The `@distinctUnionOfArrays` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfObjectsKeyValueOperator`
The `@distinctUnionOfObjects` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSDistinctUnionOfSetsKeyValueOperator`
The `@distinctUnionOfSets` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSMaximumKeyValueOperator`
The `@max` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSMinimumKeyValueOperator`
The `@min` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSSumKeyValueOperator`
The `@sum` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfArraysKeyValueOperator`
The `@unionOfArrays` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfObjectsKeyValueOperator`
The `@unionOfObjects` array operator.

Available in iOS 2.0 and later.

Declared in `NSKeyValueCoding.h`.

`NSUnionOfSetsKeyValueOperator`
The `@unionOfSets` array operator.

Available in iOS 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
Companion guide	Key-Value Observing Programming Guide

Overview

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 1600) *required method*

This message is sent to the receiver when the value at the specified key path relative to the given object has changed. (required)

Registering for Observation

`- addObserver:forKeyPath:options:context:` (page 1598) *required method*

Registers *anObserver* to receive KVO notifications for the specified key-path relative to the receiver. (required)

- `removeObserver:forKeyPath:` (page 1601) *required method*
  Stops a given object from receiving change notifications for the property specified by a given key-path relative to the receiver. (required)

Notifying Observers of Changes

- `willChangeValueForKey:` (page 1603) *required method*
  Invoked to inform the receiver that the value of a given property is about to change. (required)
- `didChangeValueForKey:` (page 1599) *required method*
  Invoked to inform the receiver that the value of a given property has changed. (required)
- `willChange:valuesAtIndexes:forKey:` (page 1602) *required method*
  Invoked to inform the receiver that the specified change is about to be executed at given indexes for a specified ordered to-many relationship. (required)
- `didChange:valuesAtIndexes:forKey:` (page 1598) *required method*
  Invoked to inform the receiver that the specified change has occurred on the indexes for a specified ordered to-many relationship. (required)
- `willChangeValueForKey:withSetMutation:usingObjects:` (page 1603) *required method*
  Invoked to inform the receiver that the specified change is about to be made to a specified unordered to-many relationship. (required)
- `didChangeValueForKey:withSetMutation:usingObjects:` (page 1599) *required method*
  Invoked to inform the receiver that the specified change was made to a specified unordered to-many relationship. (required)

Observing Customization

+ `automaticallyNotifiesObserversForKey:` (page 1596) *required method*
  Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key. (required)
+ `keyPathsForValuesAffectingValueForKey:` (page 1597) *required method*
  Returns a set of key paths for properties whose values affect the value of the specified key. (required)
- `setObservationInfo:` (page 1602) *required method*
  Sets the observation info for the receiver. (required)
- `observationInfo` (page 1600) *required method*
  Returns a pointer that identifies information about all of the observers that are registered with the receiver. (required)

Class Methods

automaticallyNotifiesObserversForKey:

Returns a Boolean value that indicates whether the receiver supports automatic key-value observation for the given key. (required)

```
+ (BOOL)automaticallyNotifiesObserversForKey:(NSString *)key
```

Return Value
`YES` if the key-value observing machinery should automatically invoke `willChangeValueForKey:` (page 1603)/`didChangeValueForKey:` (page 1599) and `willChange:valuesAtIndexes:forKey:` (page 1602)/`didChange:valuesAtIndexes:forKey:` (page 1598) 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 iOS 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. (required)

```
+ (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 iOS 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. (required)

```
- (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 1600).

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 1605).

context
Arbitrary data that is passed to *anObserver* in observeValueForKeyPath:ofObject:change:context: (page 1600).

Discussion
Neither the receiver, nor *anObserver*, are retained.

Availability
Available in iOS 2.0 and later.

See Also
`- removeObserver:forKeyPath:` (page 1601)

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. (required)

```
- (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 iOS 2.0 and later.

See Also

- `willChange:valuesAtIndexes:forKey:` (page 1602)
- `didChangeValueForKey:` (page 1599)

Declared In

`NSKeyValueObserving.h`

didChangeValueForKey:

Invoked to inform the receiver that the value of a given property has changed. (required)

```
- (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 iOS 2.0 and later.

See Also

- `willChangeValueForKey:` (page 1603)
- `didChange:valuesAtIndexes:forKey:` (page 1598)

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. (required)

```
- (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 1607)).

Discussion
You invoke this method when implementing key-value observer compliance manually.

Availability
Available in iOS 2.0 and later.

See Also
`- willChangeValueForKey:withSetMutation:usingObjects:` (page 1603)

Declared In
`NSKeyValueObserving.h`

observationInfo

Returns a pointer that identifies information about all of the observers that are registered with the receiver. (required)

```
- (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 iOS 2.0 and later.

See Also
`- setObservationInfo:` (page 1602)

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. (required)

```
- (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 1606).

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 iOS 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. (required)

```
- (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 iOS 2.0 and later.

See Also

- addObserver:forKeyPath:options:context: (page 1598)

Declared In

NSKeyValueObserving.h

setObservationInfo:

Sets the observation info for the receiver. (required)

```
- (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 iOS 2.0 and later.

See Also

`- observationInfo` (page 1600)

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. (required)

```
- (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 1598) must be invoked with the same parameters.

Availability

Available in iOS 2.0 and later.

See Also
- `didChange:valuesAtIndexes:forKey:` (page 1598)
- `willChangeValueForKey:` (page 1603)

Declared In
`NSKeyValueObserving.h`

willChangeValueForKey:

Invoked to inform the receiver that the value of a given property is about to change. (required)

```
- (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 1599) must be invoked with the same parameter.

Availability
Available in iOS 2.0 and later.

See Also
- `didChangeValueForKey:` (page 1599)
- `willChange:valuesAtIndexes:forKey:` (page 1602)

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. (required)

```
- (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 1607)).

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 1599) must be invoked with the same parameters.

Availability
Available in iOS 2.0 and later.

See Also
`- didChangeValueForKey:withSetMutation:usingObjects:` (page 1599)

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 1600) 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSKeyValueObserving.h`.

Declared In
`NSKeyValueObserving.h`

NSKeyValueObservingOptions

These constants are passed to `addObserver:forKeyPath:options:context:` (page 1598) and determine the values that are returned as part of the change dictionary passed to an `observeValueForKeyPath:ofObject:change:context:` (page 1600). 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 iOS 2.0 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueObservingOptionOld`
Indicates that the change dictionary should contain the old attribute value, if applicable.

Available in iOS 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 iOS 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 iOS 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 1600).

```
NSString *const NSKeyValueChangeKindKey;
NSString *const NSKeyValueChangeNewKey;
NSString *const NSKeyValueChangeOldKey;
NSString *const NSKeyValueChangeIndexesKey;
NSString *const NSKeyValueChangeNotificationIsPriorKey;
```

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 1603)/`didChangeValueForKey:` (page 1599) 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 1602)/`didChange:valuesAtIndexes:forKey:` (page 1598) has otherwise been invoked.

  You can use `NSNumber`'s `intValue` (page 879) method to retrieve the integer value of the change kind.

  Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSKeyValueObserving.h`.

`NSKeyValueChangeNotificationIsPriorKey`

If the value of `NSKeyValueObservingOptionPrior` was specified at observer registration time, and this notification is sent prior to a change as a result.

The change dictionary contains an `NSKeyValueChangeNotificationIsPriorKey` entry whose value is an `NSNumber` wrapping `YES`.

Available in iOS 2.0 and later.

Declared in `NSKeyValueObserving.h`.

NSKeyValueSetMutationKind

These constants are specified as the parameter to the methods `willChangeValueForKey:withSetMutation:usingObjects:` (page 1603) and `didChangeValueForKey:withSetMutation:usingObjects:` (page 1599).

```
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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSKeyValueObserving.h`.

NSLocking Protocol Reference

Adopted by	NSConditionLock NSLock NSRecursiveLock
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSLock.h
Companion guide	Threading Programming Guide

Overview

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 1609) *required method*
  Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired. (required)
- `unlock` (page 1610) *required method*
  Relinquishes a previously acquired lock. (required)

Instance Methods

lock

Attempts to acquire a lock, blocking a thread's execution until the lock can be acquired. (required)

```
- (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 1610).

Availability
Available in iOS 2.0 and later.

Declared In
`NSLock.h`

unlock

Relinquishes a previously acquired lock. (required)

```
- (void)unlock
```

Availability
Available in iOS 2.0 and later.

Declared In
`NSLock.h`

NSMachPortDelegate Protocol Reference

Conforms to	NSPortDelegate
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSPort.h
Companion guide	Distributed Objects Programming Topics

Overview

The `NSMachPortDelegate` protocol defines the optional methods implemented by delegates of `NSMachPort` objects.

Tasks

Handling Mach Messages

`- handleMachMessage:` (page 1611)
Process an incoming Mach message.

Instance 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 either `handleMachMessage:` or the `NSPortDelegate Protocol` protocol method `handlePortMessage:`.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

Declared In
`NSPort.h`

NSMutableCopying Protocol Reference

Adopted by	Various Cocoa classes
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSObject.h
Companion guide	Memory Management Programming Guide

Overview

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 1614), but mutable copying is commonly invoked with the convenience method `mutableCopy`. The `mutableCopy` method is defined for all NSObjects and simply invokes `mutableCopyWithZone:` (page 1614) with the default zone.

If a subclass inherits NSMutableCopying from its superclass and declares additional instance variables, the subclass has to override `mutableCopyWithZone:` (page 1614) to properly handle its own instance variables, invoking the superclass's implementation first.

Tasks

Copying

- `mutableCopyWithZone:` (page 1614) *required method*

  Returns a new instance that's a mutable copy of the receiver. (required)

Instance Methods

mutableCopyWithZone:

Returns a new instance that's a mutable copy of the receiver. (required)

```
- (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 1702).

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 iOS 2.0 and later.

See Also

- copyWithZone: (page 1554) (NSCopying protocol)
- mutableCopy (page 974) (NSObject class)

Declared In

NSObject.h

NSNetServiceBrowserDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	BonjourWeb

Overview

The `NSNetServiceBrowserDelegate` protocol defines the optional methods implemented by delegates of `NSNetServiceBrowser` objects.

Tasks

Using Network Service Browsers

- `netServiceBrowser:didFindDomain:moreComing:` (page 1616)
  Tells the delegate the sender found a domain.
- `netServiceBrowser:didRemoveDomain:moreComing:` (page 1617)
  Tells the delegate the a domain has disappeared or has become unavailable.
- `netServiceBrowser:didFindService:moreComing:` (page 1616)
  Tells the delegate the sender found a service.
- `netServiceBrowser:didRemoveService:moreComing:` (page 1618)
  Tells the delegate a service has disappeared or has become unavailable.
- `netServiceBrowserWillSearch:` (page 1619)
  Tells the delegate that a search is commencing.
- `netServiceBrowser:didNotSearch:` (page 1617)
  Tells the delegate that a search was not successful.
- `netServiceBrowserDidStopSearch:` (page 1618)
  Tells the delegate that a search was stopped.

Instance 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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- searchForBrowsableDomains (page 832) (NSNetServiceBrowser)
- searchForRegistrationDomains (page 833) (NSNetServiceBrowser)

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 1623) or `netService:didNotResolve:` (page 1622) delegate messages of the `NSNetService` class.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- `searchForServicesOfType:inDomain:` (page 833) (NSNetServiceBrowser)

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` (page 827) and `NSNetServicesErrorDomain` (page 827) to retrieve the error information from the dictionary.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- `netServiceBrowserWillSearch:` (page 1619)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 834) message from its client, *netServiceBrowser* sends a `netServiceBrowserDidStopSearch:` message to its delegate. The delegate then performs any necessary cleanup.

Availability

Available in iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

`- stop` (page 834) (NSNetServiceBrowser)

Declared In

`NSNetServices.h`

netServiceBrowserWillSearch:

Tells the delegate that a search 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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

`- netServiceBrowser:didNotSearch:` (page 1617)

Declared In

`NSNetServices.h`

NSNetServiceDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSNetServices.h
Companion guides	Bonjour Overview NSNetServices and CFNetServices Programming Guide
Related sample code	BonjourWeb

Overview

The `NSNetServiceDelegate` protocol defines the optional methods implemented by delegates of `NSNetService` objects.

Tasks

Using Network Services

- `netServiceWillPublish:` (page 1624)
  Notifies the delegate that the network is ready to publish the service.
- `netService:didNotPublish:` (page 1622)
  Notifies the delegate that a service could not be published.
- `netServiceDidPublish:` (page 1623)
  Notifies the delegate that a service was successfully published.
- `netServiceWillResolve:` (page 1625)
  Notifies the delegate that the network is ready to resolve the service.
- `netService:didNotResolve:` (page 1622)
  Informs the delegate that an error occurred during resolution of a given service.
- `netServiceDidResolveAddress:` (page 1623)
  Informs the delegate that the address for a given service was resolved.
- `netService:didUpdateTXTRecordData:` (page 1623)
  Notifies the delegate that the TXT record for a given service has been updated.

- `netServiceDidStop:` (page 1624)
  Informs the delegate that a `publish` (page 821) or `resolveWithTimeout:` (page 823) request was stopped.

Instance 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 1624) message has been delivered to the delegate.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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` (page 827) and `NSNetServicesErrorDomain` (page 827).

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- startMonitoring (page 825) (NSNetService)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 817) method to retrieve the service's address.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
`- addresses` (page 817) (NSNetService)

Declared In
`NSNetServices.h`

netServiceDidStop:

Informs the delegate that a `publish` (page 821) or `resolveWithTimeout:` (page 823) request was stopped.

```
- (void)netServiceDidStop:(NSNetService *)sender
```

Parameters
sender
The service that stopped.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 1622) method if an error occurs.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 1622) method if an error occurs.

Availability

Available in iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

Declared In

`NSNetServices.h`

NSObject Protocol Reference

Adopted by	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSObject.h
Companion guides	Cocoa Fundamentals Guide Memory Management Programming Guide

Overview

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 1638), release (page 1636), and autorelease (page 1629) methods—can also integrate with the object management and deallocation scheme defined in Foundation (for more information see, for example, *Memory Management Programming Guide*). 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 1630) *required method*

Returns the class object for the receiver's class. (required)

- `superclass` (page 1640) *required method*
  Returns the class object for the receiver's superclass. (required)

Identifying and Comparing Objects

- `isEqual:` (page 1632) *required method*
  Returns a Boolean value that indicates whether the receiver and a given object are equal. (required)
- `hash` (page 1631) *required method*
  Returns an integer that can be used as a table address in a hash table structure. (required)
- `self` (page 1639) *required method*
  Returns the receiver. (required)

Managing Reference Counts

- `retain` (page 1638) *required method*
  Increments the receiver's reference count. (required)
- `release` (page 1636) *required method*
  Decrements the receiver's reference count. (required)
- `autorelease` (page 1629) *required method*
  Adds the receiver to the current autorelease pool. (required)
- `retainCount` (page 1638) *required method*
  Returns the receiver's reference count. (required)

Testing Object Inheritance, Behavior, and Conformance

- `isKindOfClass:` (page 1632) *required method*
  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. (required)
- `isMemberOfClass:` (page 1633) *required method*
  Returns a Boolean value that indicates whether the receiver is an instance of a given class. (required)
- `respondsToSelector:` (page 1637) *required method*
  Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message. (required)
- `conformsToProtocol:` (page 1630) *required method*
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol. (required)

Describing Objects

- `description` (page 1631) *required method*
  Returns a string that describes the contents of the receiver. (required)

Sending Messages

- `performSelector:` (page 1634) *required method*
  Sends a specified message to the receiver and returns the result of the message. (required)
- `performSelector:withObject:` (page 1635) *required method*
  Sends a message to the receiver with an object as the argument. (required)
- `performSelector:withObject:withObject:` (page 1635) *required method*
  Sends a message to the receiver with two objects as arguments. (required)

Determining Allocation Zones

- `zone` (page 1640) *required method*
  Returns a pointer to the zone from which the receiver was allocated. (required)

Identifying Proxies

- `isProxy` (page 1634) *required method*
  Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`. (required)

Instance Methods

autorelease

Adds the receiver to the current autorelease pool. (required)

```
- (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*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

Availability
Available in iOS 2.0 and later.

See Also
- `retain` (page 1638)

Related Sample Code
BonjourWeb

CryptoExercise
GKTank
MultipleDetailViews
ScrollViewSuite

Declared In
`NSObject.h`

class

Returns the class object for the receiver's class. (required)

```
- (Class)class
```

Return Value
The class object for the receiver's class.

Availability
Available in iOS 2.0 and later.

See Also
`class` (page 952) (`NSObject` class)

Declared In
`NSObject.h`

conformsToProtocol:

Returns a Boolean value that indicates whether the receiver conforms to a given protocol. (required)

```
- (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 953) 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 iOS 2.0 and later.

Declared In
`NSObject.h`

description

Returns a string that describes the contents of the receiver. (required)

```
- (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 iOS 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. (required)

```
- (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 1632) 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 iOS 2.0 and later.

See Also
`- isEqual:` (page 1632)

Related Sample Code
CryptoExercise

Declared In
`NSObject.h`

isEqual:

Returns a Boolean value that indicates whether the receiver and a given object are equal. (required)

```
- (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 1631) in your subclass.

Availability

Available in iOS 2.0 and later.

See Also

- hash (page 1631)

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. (required)

```
- (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 iOS 2.0 and later.

See Also
`- isMemberOfClass:` (page 1633)

Declared In
`NSObject.h`

isMemberOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of a given class. (required)

```
- (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 iOS 2.0 and later.

See Also
- `isKindOfClass:` (page 1632)

Declared In
`NSObject.h`

isProxy

Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`. (required)

```
- (BOOL)isProxy
```

Return Value
`NO` if the receiver really descends from `NSObject`, otherwise `YES`.

Discussion
This method is necessary because sending `isKindOfClass:` (page 1632) or `isMemberOfClass:` (page 1633) 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 iOS 2.0 and later.

Declared In
`NSObject.h`

performSelector:

Sends a specified message to the receiver and returns the result of the message. (required)

```
- (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 iOS 2.0 and later.

See Also
- performSelector:withObject: (page 1635)
- performSelector:withObject:withObject: (page 1635)

Declared In
NSObject.h

performSelector:withObject:

Sends a message to the receiver with an object as the argument. (required)

```
- (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 1634) 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 iOS 2.0 and later.

See Also
- performSelector:withObject:withObject: (page 1635)
methodForSelector: (page 973) (NSObject class)

Declared In
NSObject.h

performSelector:withObject:withObject:

Sends a message to the receiver with two objects as arguments. (required)

```
- (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 1634) 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 iOS 2.0 and later.

See Also

- performSelector:withObject: (page 1635)

methodForSelector: (page 973) (NSObject class)

Declared In

NSObject.h

release

Decrements the receiver's reference count. (required)

```
- (oneway void)release
```

Discussion

The receiver is sent a dealloc (page 966) 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*.

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 iOS 2.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise
GKRocket
ScrollViewSuite
SpeakHere

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. (required)

```
- (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 957) 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 iOS 2.0 and later.

See Also
`forwardInvocation:` (page 970) (`NSObject` class)
`instancesRespondToSelector:` (page 957) (`NSObject` class)

Declared In
`NSObject.h`

retain

Increments the receiver's reference count. (required)

```
- (id)retain
```

Return Value
`self`.

Discussion
You send an object a `retain` message when you want to prevent it from being deallocated until you have finished using it.

An object is deallocated automatically when its reference count reaches 0. `retain` messages increment the reference count, and `release` (page 1636) messages decrement it. For more information on this mechanism, see *Memory Management Programming Guide*.

As a convenience, `retain` returns `self` because it may be used in nested expressions.

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 iOS 2.0 and later.

See Also
- `autorelease` (page 1629)
- `release` (page 1636)

Related Sample Code
BonjourWeb
CryptoExercise
GKRocket
SpeakHere
WiTap

Declared In
`NSObject.h`

retainCount

Returns the receiver's reference count. (required)

```
- (NSUInteger)retainCount
```

Return Value
The receiver's reference count.

Discussion
You might override this method in a class to implement your own reference-counting scheme. For objects that never get released (that is, their `release` (page 1636) 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 1629) messages sent to the receiver.

Important: This method is typically of no value in debugging memory management issues. Because any number of framework objects may have retained an object in order to hold references to it, while at the same time autorelease pools may be holding any number of deferred releases on an object, it is very unlikely that you can get useful information from this method.

To understand the fundamental rules of memory management that you must abide by, read "Memory Management Rules". To diagnose memory management problems, use a suitable tool:

- The LLVM/Clang Static analyzer can typically find memory management problems even before you run your program.
- The Object Alloc instrument in the Instruments application (see *Instruments User Guide*) can track object allocation and destruction.
- Shark (see *Shark User Guide*) also profiles memory allocations (amongst numerous other aspects of your program).

Special Considerations

If garbage collection is enabled, the return value is undefined.

Availability
Available in iOS 2.0 and later.

See Also
- `autorelease` (page 1629)
- `retain` (page 1638)

Related Sample Code
CryptoExercise

Declared In
`NSObject.h`

self

Returns the receiver. (required)

```
- (id)self
```

Return Value
The receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- class` (page 1630)

Related Sample Code
BonjourWeb
KeyboardAccessory

Declared In
`NSObject.h`

superclass

Returns the class object for the receiver's superclass. (required)

```
- (Class)superclass
```

Return Value
The class object for the receiver's superclass.

Availability
Available in iOS 2.0 and later.

See Also
`superclass` (page 962) (`NSObject` class)

Declared In
`NSObject.h`

zone

Returns a pointer to the zone from which the receiver was allocated. (required)

```
- (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 iOS 2.0 and later.

See Also
`allocWithZone:` (page 950) (`NSObject` class)

Declared In
`NSObject.h`

NSPortDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSPort.h
Companion guides	Run Loops Distributed Objects Programming Topics

Overview

The `NSPortDelegate` protocol defines the optional methods implemented by delegates of `NSPort` objects.

Tasks

Handling Port Messages

`- handlePortMessage:` (page 1641)
Processes a given incoming message on the port.

Instance Methods

handlePortMessage:

Processes a given incoming message on the port.

```
- (void)handlePortMessage:(NSPortMessage *)portMessage
```

Parameters

portMessage
An incoming port message.

Discussion
See *NSPort Class Reference* for more information.

The delegate should implement either `handlePortMessage:` or the `NSMachPortDelegate Protocol` protocol method `handleMachMessage:` (page 1611). You must not implement both delegate methods.

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

Declared In
`NSPort.h`

NSStreamDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSStream.h
Companion guide	Stream Programming Guide for Cocoa

Overview

The `NSStreamDelegate` protocol defines the optional methods implemented by delegates of `NSStream` objects.

Tasks

Using Streams

`- stream:handleEvent:` (page 1643)
The delegate receives this message when a given event has occurred on a given stream.

Instance 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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

Declared In
NSStream.h

NSURLAuthenticationChallengeSender Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLAuthenticationChallenge.h
Companion guide	URL Loading System Programming Guide

Overview

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 1428) or `download:didReceiveAuthenticationChallenge:`. The different methods provide different ways of responding to authentication challenges.

Tasks

Protocol Methods

- `cancelAuthenticationChallenge:` (page 1646) *required method*
  Cancels a given authentication challenge. (required)
- `continueWithoutCredentialForAuthenticationChallenge:` (page 1646) *required method*
  Attempt to continue downloading a request without providing a credential for a given challenge. (required)
- `useCredential:forAuthenticationChallenge:` (page 1646) *required method*
  Attempt to use a given credential for a given authentication challenge. (required)

Instance Methods

cancelAuthenticationChallenge:

Cancels a given authentication challenge. (required)

```
- (void)cancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

challenge

The authentication challenge to cancel.

Availability

Declared In

NSURLAuthenticationChallenge.h

continueWithoutCredentialForAuthenticationChallenge:

Attempt to continue downloading a request without providing a credential for a given challenge. (required)

```
- (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

Declared In

NSURLAuthenticationChallenge.h

useCredential:forAuthenticationChallenge:

Attempt to use a given credential for a given authentication challenge. (required)

```
- (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

Declared In
NSURLAuthenticationChallenge.h

NSURLProtocolClient Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	Foundation/NSURLProtocol.h
Companion guide	URL Loading System Programming Guide

Overview

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 1650) *required method*
  Sent to indicate to the URL loading system that a cached response is valid. (required)
- `URLProtocol:didCancelAuthenticationChallenge:` (page 1650) *required method*
  Sent to indicate to the URL loading system that an authentication challenge has been canceled. (required)
- `URLProtocol:didFailWithError:` (page 1650) *required method*
  Sent when the load request fails due to an error. (required)
- `URLProtocol:didLoadData:` (page 1651) *required method*
  An NSURLProtocol subclass instance, *`protocol`*, sends this message to `[protocol client]` as it loads *`data`*. (required)
- `URLProtocol:didReceiveAuthenticationChallenge:` (page 1651) *required method*
  Sent to indicate to the URL loading system that an authentication challenge has been received. (required)
- `URLProtocol:didReceiveResponse:cacheStoragePolicy:` (page 1652) *required method*
  Sent to indicate to the URL loading system that the protocol implementation has created a response object for the request. (required)

- URLProtocol:wasRedirectedToRequest:redirectResponse: (page 1652) *required method*
  Sent to indicate to the URL loading system that the protocol implementation has been redirected. (required)
- URLProtocolDidFinishLoading: (page 1652) *required method*
  Sent to indicate to the URL loading system that the protocol implementation has finished loading. (required)

Instance Methods

URLProtocol:cachedResponseIsValid:

Sent to indicate to the URL loading system that a cached response is valid. (required)

```
- (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

Declared In
NSURLProtocol.h

URLProtocol:didCancelAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been canceled. (required)

```
- (void)URLProtocol:(NSURLProtocol *)protocol
    didCancelAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
```

Parameters

protocol
The URL protocol object sending the message.

challenge
The authentication challenge that was canceled.

Availability

Declared In
NSURLProtocol.h

URLProtocol:didFailWithError:

Sent when the load request fails due to an error. (required)

```
- (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

Declared In
NSURLProtocol.h

URLProtocol:didLoadData:

An NSURLProtocol subclass instance, *protocol*, sends this message to `[protocol client]` as it loads *data*. (required)

```
- (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

Declared In
NSURLProtocol.h

URLProtocol:didReceiveAuthenticationChallenge:

Sent to indicate to the URL loading system that an authentication challenge has been received. (required)

```
- (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

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. (required)

```
- (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

Declared In
NSURLProtocol.h

URLProtocol:wasRedirectedToRequest:redirectResponse:

Sent to indicate to the URL loading system that the protocol implementation has been redirected. (required)

```
- (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

Declared In
NSURLProtocol.h

URLProtocolDidFinishLoading:

Sent to indicate to the URL loading system that the protocol implementation has finished loading. (required)

```
- (void)URLProtocolDidFinishLoading:(NSURLProtocol *)protocol
```

Parameters

protocol

The URL protocol object sending the message.

Availability

Declared In

NSURLProtocol.h

NSXMLParserDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	Foundation/NSXMLParser.h
Companion guide	Event-Driven XML Programming Guide

Overview

The `NSXMLParserDelegate` protocol defines the optional methods implemented by delegates of `NSXMLParser` objects.

Tasks

Handling XML

- `parserDidStartDocument:` (page 1666)
  Sent by the parser object to the delegate when it begins parsing a document.
- `parserDidEndDocument:` (page 1666)
  Sent by the parser object to the delegate when it has successfully completed parsing.
- `parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 1657)
  Sent by a parser object to its delegate when it encounters a start tag for a given element.
- `parser:didStartMappingPrefix:toURI:` (page 1658)
  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 1657)
  Sent by a parser object to its delegate when a given namespace prefix goes out of scope.
- `parser:resolveExternalEntityName:systemID:` (page 1665)
  Sent by a parser object to its delegate when it encounters a given external entity with a specific system ID.
- `parser:parseErrorOccurred:` (page 1664)
  Sent by a parser object to its delegate when it encounters a fatal error.

- `parser:validationErrorOccurred:` (page 1666)
  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 1660)
  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 1662)
  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 1663)
  Sent by a parser object to its delegate when it encounters a processing instruction.
- `parser:foundComment:` (page 1660)
  Sent by a parser object to its delegate when it encounters a comment in the XML.
- `parser:foundCDATA:` (page 1659)
  Sent by a parser object to its delegate when it encounters a CDATA block.
- `parser:didEndElement:namespaceURI:qualifiedName:` (page 1656) Deprecated in iOS 2.0
  Sent by a parser object to its delegate when it encounters an end tag for a specific element.

Handling the DTD

- `parser:foundAttributeDeclarationWithName:forElement:type:defaultValue:` (page 1659)
  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 1661)
  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 1661)
  Sent by a parser object to its delegate when it encounters an external entity declaration.
- `parser:foundInternalEntityDeclarationWithName:value:` (page 1662)
  Sent by a parser object to the delegate when it encounters an internal entity declaration.
- `parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1664)
  Sent by a parser object to its delegate when it encounters an unparsed entity declaration.
- `parser:foundNotationDeclarationWithName:publicID:systemID:` (page 1663)
  Sent by a parser object to its delegate when it encounters a notation declaration.

Instance 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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

`- parser:didStartElement:namespaceURI:qualifiedName:attributes:` (page 1657)

`setShouldProcessNamespaces:` (page 1531) (NSXMLParser)

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 1531) method.

Availability

Available in iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

`- parser:didStartMappingPrefix:toURI:` (page 1658)

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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

`- parser:didEndElement:namespaceURI:qualifiedName:` (page 1656)

`setShouldProcessNamespaces:` (page 1531) (NSXMLParser)

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 1531) method.

Availability

Available in iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:didEndMappingPrefix: (page 1657)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 1657)

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 1238).

Availability
Available in iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:didStartElement:namespaceURI:qualifiedName:attributes: (page 1657)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- `parser:foundInternalEntityDeclarationWithName:value:` (page 1662)
- `parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1664)
- `parser:resolveExternalEntityName:systemID:` (page 1665)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- `parser:foundCharacters:` (page 1660)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- `parser:foundExternalEntityDeclarationWithName:publicID:systemID:` (page 1661)
- `parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName:` (page 1664)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 1661)
- parser:foundInternalEntityDeclarationWithName:value: (page 1662)
- parser:resolveExternalEntityName:systemID: (page 1665)

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 *parser* a parserError (page 1529) message. You can also send the parser lineNumber (page 1529) and columnNumber (page 1527) 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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:validationErrorOccurred: (page 1666)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
- parser:foundExternalEntityDeclarationWithName:publicID:systemID: (page 1661)
- parser:foundUnparsedEntityDeclarationWithName:publicID:systemID:notationName: (page 1664)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
`- parser:parseErrorOccurred:` (page 1664)

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 iOS 2.0 and later.
Available as part of an informal protocol prior to iOS 4.0.

See Also
`- parserDidStartDocument:` (page 1666)

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 iOS 2.0 and later.

Available as part of an informal protocol prior to iOS 4.0.

See Also

- parserDidEndDocument: (page 1666)

Declared In

NSXMLParser.h

Functions

Foundation Functions Reference

Framework: Foundation/Foundation.h

Overview

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 Programming Guide*.

`NSAssert` (page 1679)
Generates an assertion if a given condition is false.

`NSAssert1` (page 1680)
Generates an assertion if a given condition is false.

`NSAssert2` (page 1681)
Generates an assertion if a given condition is false.

`NSAssert3` (page 1682)
Generates an assertion if a given condition is false.

`NSAssert4` (page 1683)
Generates an assertion if a given condition is false.

`NSAssert5` (page 1684)
Generates an assertion if a given condition is false.

`NSCAssert` (page 1685)
Generates an assertion if the given condition is false.

`NSCAssert1` (page 1685)
`NSCAssert1` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert2` (page 1686)
`NSCAssert2` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert3` (page 1687)
`NSCAssert3` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert4` (page 1688)
`NSCAssert4` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert5` (page 1688)
`NSCAssert5` is one of a series of macros that generate assertions if the given condition is false.

`NSCParameterAssert` (page 1693)
Evaluates the specified parameter.

`NSParameterAssert` (page 1712)
Validates the specified parameter.

Bundles

For additional information on generating strings files see Strings Files in *Internationalization Programming Topics*.

`NSLocalizedString` (page 1706)
Returns a localized version of a string.

`NSLocalizedStringFromTable` (page 1707)
Returns a localized version of a string.

`NSLocalizedStringWithDefaultValue` (page 1708)
Returns a localized version of a string.

`NSLocalizedStringFromTableInBundle` (page 1707) Deprecated in iOS 4.0
Returns a localized version of a string.

Byte Ordering

`NSConvertHostDoubleToSwapped` (page 1690)
Performs a type conversion.

`NSConvertHostFloatToSwapped` (page 1690)
Performs a type conversion.

`NSConvertSwappedDoubleToHost` (page 1691)
Performs a type conversion.

`NSConvertSwappedFloatToHost` (page 1691)
Performs a type conversion.

`NSHostByteOrder` (page 1705)
Returns the endian format.

`NSSwapBigDoubleToHost` (page 1720)
A utility for swapping the bytes of a number.

`NSSwapBigFloatToHost` (page 1720)
A utility for swapping the bytes of a number.

`NSSwapBigIntToHost` (page 1721)
A utility for swapping the bytes of a number.

`NSSwapBigLongLongToHost` (page 1721)
A utility for swapping the bytes of a number.

`NSSwapBigLongToHost` (page 1721)
A utility for swapping the bytes of a number.

NSSwapBigShortToHost (page 1722)
A utility for swapping the bytes of a number.

NSSwapDouble (page 1722)
A utility for swapping the bytes of a number.

NSSwapFloat (page 1723)
A utility for swapping the bytes of a number.

NSSwapHostDoubleToBig (page 1723)
A utility for swapping the bytes of a number.

NSSwapHostDoubleToLittle (page 1724)
A utility for swapping the bytes of a number.

NSSwapHostFloatToBig (page 1724)
A utility for swapping the bytes of a number.

NSSwapHostFloatToLittle (page 1724)
A utility for swapping the bytes of a number.

NSSwapHostIntToBig (page 1725)
A utility for swapping the bytes of a number.

NSSwapHostIntToLittle (page 1725)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToBig (page 1726)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToLittle (page 1726)
A utility for swapping the bytes of a number.

NSSwapHostLongToBig (page 1727)
A utility for swapping the bytes of a number.

NSSwapHostLongToLittle (page 1727)
A utility for swapping the bytes of a number.

NSSwapHostShortToBig (page 1727)
A utility for swapping the bytes of a number.

NSSwapHostShortToLittle (page 1728)
A utility for swapping the bytes of a number.

NSSwapInt (page 1728)
A utility for swapping the bytes of a number.

NSSwapLittleDoubleToHost (page 1729)
A utility for swapping the bytes of a number.

NSSwapLittleFloatToHost (page 1729)
A utility for swapping the bytes of a number.

NSSwapLittleIntToHost (page 1730)
A utility for swapping the bytes of a number.

NSSwapLittleLongLongToHost (page 1730)
A utility for swapping the bytes of a number.

NSSwapLittleLongToHost (page 1730)
A utility for swapping the bytes of a number.

NSSwapLittleShortToHost (page 1731)
A utility for swapping the bytes of a number.

`NSSwapLong` (page 1731)
A utility for swapping the bytes of a number.

`NSSwapLongLong` (page 1732)
A utility for swapping the bytes of a number.

`NSSwapShort` (page 1732)
A utility for swapping the bytes of a number.

Decimals

You can also use the class `NSDecimalNumber` for decimal arithmetic.

`NSDecimalAdd` (page 1695)
Adds two decimal values.

`NSDecimalCompact` (page 1695)
Compacts the decimal structure for efficiency.

`NSDecimalCompare` (page 1696)
Compares two decimal values.

`NSDecimalCopy` (page 1696)
Copies the value of a decimal number.

`NSDecimalDivide` (page 1696)
Divides one decimal value by another.

`NSDecimalIsNotANumber` (page 1697)
Returns a Boolean that indicates whether a given decimal contains a valid number.

`NSDecimalMultiply` (page 1697)
Multiplies two decimal numbers together.

`NSDecimalMultiplyByPowerOf10` (page 1698)
Multiplies a decimal by the specified power of 10.

`NSDecimalNormalize` (page 1698)
Normalizes the internal format of two decimal numbers to simplify later operations.

`NSDecimalPower` (page 1699)
Raises the decimal value to the specified power.

`NSDecimalRound` (page 1700)
Rounds off the decimal value.

`NSDecimalString` (page 1700)
Returns a string representation of the decimal value.

`NSDecimalSubtract` (page 1701)
Subtracts one decimal value from another.

Exception Handling

You can find the following macros implemented in `NSException.h`. *Exception Programming Topics* 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 in *The Objective-C Programming Language*.

NS_DURING (page 1737)
Marks the start of the exception-handling domain.

NS_ENDHANDLER (page 1737)
Marks the end of the local event handler.

NS_HANDLER (page 1738)
Marks the end of the exception-handling domain and the start of the local exception handler.

NS_VALUERETURN (page 1738)
Permits program control to exit from an exception-handling domain with a value of a specified type.

NS_VOIDRETURN (page 1739)
Permits program control to exit from an exception-handling domain.

Managing Object Allocation and Deallocation

NSAllocateObject (page 1678)
Creates and returns a new instance of a given class.

NSCopyObject (page 1692)
Creates an exact copy of an object.

NSDeallocateObject (page 1694)
Destroys an existing object.

NSDecrementExtraRefCountWasZero (page 1701)
Decrements the specified object's reference count.

NSExtraRefCount (page 1702)
Returns the specified object's reference count.

NSIncrementExtraRefCount (page 1705)
Increments the specified object's reference count.

NSShouldRetainWithZone (page 1717)
Indicates whether an object should be retained.

Interacting with the Objective-C Runtime

NSGetSizeAndAlignment (page 1703)
Obtains the actual size and the aligned size of an encoded type.

NSClassFromString (page 1689)
Obtains a class by name.

NSStringFromClass (page 1718)
Returns the name of a class as a string.

NSSelectorFromString (page 1716)
Returns the selector with a given name.

NSStringFromSelector (page 1719)
Returns a string representation of a given selector.

NSStringFromProtocol (page 1719)
Returns the name of a protocol as a string.

NSProtocolFromString (page 1713)
Returns a the protocol with a given name.

Logging Output

NSLog (page 1709)
Logs an error message to the Apple System Log facility.

NSLogv (page 1709)
Logs an error message to the Apple System Log facility.

Managing File Paths

NSFullUserName (page 1703)
Returns a string containing the full name of the current user.

NSHomeDirectory (page 1704)
Returns the path to the current user's home directory.

NSHomeDirectoryForUser (page 1704)
Returns the path to a given user's home directory.

NSOpenStepRootDirectory (page 1712)
Returns the root directory of the user's system.

NSSearchPathForDirectoriesInDomains (page 1715)
Creates a list of directory search paths.

NSTemporaryDirectory (page 1733)
Returns the path of the temporary directory for the current user.

NSUserName (page 1734)
Returns the logon name of the current user.

Managing Ranges

NSEqualRanges (page 1702)
Returns a Boolean value that indicates whether two given ranges are equal.

NSIntersectionRange (page 1706)
Returns the intersection of the specified ranges.

NSLocationInRange (page 1708)
Returns a Boolean value that indicates whether a specified position is in a given range.

NSMakeRange (page 1711)
Creates a new NSRange from the specified values.

NSMaxRange (page 1711)
Returns the sum of the location and length of the range.

NSRangeFromString (page 1713)
Returns a range from a textual representation.

NSStringFromRange (page 1719)
Returns a string representation of a range.

NSUnionRange (page 1733)
Returns the union of the specified ranges.

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 1704)
Returns the top-level error handler.

NSSetUncaughtExceptionHandler (page 1717)
Changes the top-level error handler.

Managing Memory

NSDefaultMallocZone (page 1702)
Returns the default zone.

NSMakeCollectable (page 1710)
Makes a newly allocated Core Foundation object eligible for collection.

NSAllocateMemoryPages (page 1678)
Allocates a new block of memory.

NSCopyMemoryPages (page 1691)
Copies a block of memory.

NSDeallocateMemoryPages (page 1694)
Deallocates the specified block of memory.

NSLogPageSize (page 1709)
Returns the binary log of the page size.

NSPageSize (page 1712)
Returns the number of bytes in a page.

NSRealMemoryAvailable (page 1714)
Returns information about the user's system.

NSRoundDownToMultipleOfPageSize (page 1715)
Returns the specified number of bytes rounded down to a multiple of the page size.

NSRoundUpToMultipleOfPageSize (page 1715)
Returns the specified number of bytes rounded up to a multiple of the page size.

Managing Zones

NSCreateZone (page 1693)
Creates a new zone.

NSRecycleZone (page 1714)
Frees memory in a zone.

NSSetZoneName (page 1717)
Sets the name of the specified zone.

`NSZoneCalloc` (page 1734)
Allocates memory in a zone.

`NSZoneFree` (page 1735)
Deallocates a block of memory in the specified zone.

`NSZoneFromPointer` (page 1735)
Gets the zone for a given block of memory.

`NSZoneMalloc` (page 1735)
Allocates memory in a zone.

`NSZoneName` (page 1736)
Returns the name of the specified zone.

`NSZoneRealloc` (page 1736)
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 iOS 2.0 and later.

See Also

`NSCopyMemoryPages` (page 1691)
`NSDeallocateMemoryPages` (page 1694)

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 iOS 2.0 and later.

See Also
`NSDeallocateObject` (page 1694)

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert1 (page 1680)
NSCAssert (page 1685)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)
NSAssert2 (page 1681)
NSAssert3 (page 1682)
NSAssert4 (page 1683)
NSAssert5 (page 1684)

NSCAssert (page 1685)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)
NSAssert1 (page 1680)
NSAssert3 (page 1682)
NSAssert4 (page 1683)
NSAssert5 (page 1684)
NSCAssert (page 1685)
NSCParameterAssert (page 1693)

NSParameterAssert (page 1712)

Declared In
NSException.h

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)
NSAssert1 (page 1680)
NSAssert2 (page 1681)
NSAssert4 (page 1683)
NSAssert5 (page 1684)
NSCAssert (page 1685)
NSCParameterAssert (page 1693)

NSParameterAssert (page 1712)

Declared In
NSException.h

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)
NSAssert1 (page 1680)
NSAssert2 (page 1681)
NSAssert3 (page 1682)
NSAssert5 (page 1684)

NSCAssert (page 1685)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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 87) 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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)

NSAssert1 (page 1680)
NSAssert2 (page 1681)
NSAssert3 (page 1682)
NSAssert4 (page 1683)
NSCAssert (page 1685)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSAssert (page 1679)
NSCAssert1 (page 1685)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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`.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSCAssert` (page 1685)
`NSCAssert2` (page 1686)
`NSCAssert3` (page 1687)
`NSCAssert4` (page 1688)
`NSCAssert5` (page 1688)
`NSCParameterAssert` (page 1693)
`NSParameterAssert` (page 1712)

Declared In
`NSException.h`

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`.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSCAssert` (page 1685)
`NSCAssert1` (page 1685)
`NSCAssert3` (page 1687)
`NSCAssert4` (page 1688)
`NSCAssert5` (page 1688)
`NSCParameterAssert` (page 1693)
`NSParameterAssert` (page 1712)

Declared In
`NSException.h`

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`.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSCAssert` (page 1685)
`NSCAssert1` (page 1685)
`NSCAssert2` (page 1686)
`NSCAssert4` (page 1688)
`NSCAssert5` (page 1688)

NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 1709)
NSLogv (page 1709)
NSCAssert (page 1685)
NSCAssert1 (page 1685)
NSCAssert2 (page 1686)
NSCAssert3 (page 1687)
NSCAssert5 (page 1688)
NSCParameterAssert (page 1693)
NSParameterAssert (page 1712)

Declared In
NSException.h

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`.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSCAssert` (page 1685)
`NSCAssert1` (page 1685)
`NSCAssert2` (page 1686)
`NSCAssert3` (page 1687)
`NSCAssert4` (page 1688)
`NSCParameterAssert` (page 1693)
`NSParameterAssert` (page 1712)

Declared In
`NSException.h`

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 iOS 2.0 and later.

See Also
NSStringFromClass (page 1718)
NSProtocolFromString (page 1713)
NSSelectorFromString (page 1716)

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 iOS 2.0 and later.

See Also
NSSwapHostDoubleToBig (page 1723)
NSSwapHostDoubleToLittle (page 1724)

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 iOS 2.0 and later.

See Also
NSSwapHostFloatToBig (page 1724)
NSSwapHostFloatToLittle (page 1724)

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 iOS 2.0 and later.

See Also
`NSSwapBigDoubleToHost` (page 1720)
`NSSwapLittleDoubleToHost` (page 1729)

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 iOS 2.0 and later.

See Also
`NSSwapBigFloatToHost` (page 1720)
`NSSwapLittleFloatToHost` (page 1729)

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 iOS 2.0 and later.

See Also
NSAllocateMemoryPages (page 1678)
NSDeallocateMemoryPages (page 1694)

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.

Special Considerations

This function is dangerous and very difficult to use correctly. It's use as part of copyWithZone: (page 954) by any class that can be subclassed, is highly error prone. Under GC or when using Objective-C 2.0, the zone is completely ignored.

This function is likely to be deprecated after Mac OS X 10.6.

Availability
Available in iOS 2.0 and later.

See Also
NSAllocateObject (page 1678)
NSDeallocateObject (page 1694)

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`.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSAssert` (page 1679)
`NSCAssert` (page 1685)
`NSParameterAssert` (page 1712)

Declared In
NSException.h

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 iOS 2.0 and later.

See Also
NSDefaultMallocZone (page 1702)
NSRecycleZone (page 1714)
NSSetZoneName (page 1717)

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 iOS 2.0 and later.

See Also
NSCopyMemoryPages (page 1691)
NSAllocateMemoryPages (page 1678)

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 iOS 2.0 and later.

See Also
NSAllocateObject (page 1678)

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 1556) in NSDecimalNumberBehaviors.

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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 1695).

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*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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 1695).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 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 1695).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 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 1695) and `NSDecimalSubtract` (page 1701) 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 1695).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 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 1695).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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*.

Availability
Available in iOS 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 1695).

For more information, see *Number and Value Programming Topics*.

Availability

Available in iOS 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 1638) or `release` (page 1636) methods.

Availability

Available in iOS 2.0 and later.

See Also

`NSExtraRefCount` (page 1702)

`NSIncrementExtraRefCount` (page 1705)

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 iOS 2.0 and later.

See Also
`NSCreateZone` (page 1693)

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 iOS 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 1705) and `NSDecrementExtraRefCountWasZero` (page 1701) in situations where you need to override an object's `retain` (page 1638) and `release` (page 1636) methods.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`NSUserName` (page 1734)

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 iOS 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 iOS 2.0 and later.

See Also
`NSSetUncaughtExceptionHandler` (page 1717)

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 iOS 2.0 and later.

See Also
`NSFullUserName` (page 1703)
`NSUserName` (page 1734)
`NSHomeDirectoryForUser` (page 1704)

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 iOS 2.0 and later.

See Also
NSFullUserName (page 1703)
NSUserName (page 1734)
NSHomeDirectory (page 1704)

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 iOS 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 iOS 2.0 and later.

See Also
NSExtraRefCount (page 1702)
NSDecrementExtraRefCountWasZero (page 1701)

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 iOS 2.0 and later.

See Also
NSUnionRange (page 1733)

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 131) on the main bundle and a nil table.

Discussion
You can specify Unicode characters in *key* using \\Uxxxx—see the -u option for for the genstrings utility.

For more information, see NSBundle.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the genstrings utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AddMusic
BonjourWeb

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 131) on the main bundle, passing it the specified *key* and *tableName*.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in iOS 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 131) on *bundle*, passing it the specified *key* and *tableName*.

Discussion
You can specify Unicode characters in *key* using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in iOS 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 131) 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`.

Special Considerations

In Mac OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the *key* parameter must not contain any high-ASCII characters.

Availability
Available in iOS 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 *loc* lies within *range*—that is, if it's greater than or equal to `range.location` and less than `range.location` plus `range.length`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSRange.h`

NSLog

Logs an error message to the Apple System Log facility.

```
void NSLog (
   NSString *format,
   ...
);
```

Discussion
Simply calls `NSLogv` (page 1709), passing it a variable number of arguments.

Availability
Available in iOS 2.0 and later.

See Also
`NSLogv` (page 1709)

Related Sample Code
AddMusic
CryptoExercise
GKRocket
ScrollViewSuite
WiTap

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 iOS 2.0 and later.

See Also
`NSRoundDownToMultipleOfPageSize` (page 1715)
`NSRoundUpToMultipleOfPageSize` (page 1715)
`NSPageSize` (page 1712)

Declared In
`NSZone.h`

NSLogv

Logs an error message to the Apple System Log facility.

```
void NSLogv (
   NSString *format,
   va_list args
);
```

Discussion
Logs an error message to the Apple System Log facility (see `man 3 asl`). If the `STDERR_FILENO` file descriptor has been redirected away from the default or is going to a tty, it will also be written there. If you want to direct output elsewhere, you need to use a custom logging facility.

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 1709) 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.

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 iOS 2.0 and later.

See Also
`NSLog` (page 1709)

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.

Availability
Available in iOS 2.0 and later.

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 iOS 2.0 and later.

Declared In
`NSRange.h`

NSMaxRange

Returns the sum of the location and length of the range.

```
NSUInteger NSMaxRange (
   NSRange range
);
```

Return Value
The sum of the location and length of the range—that is, `range.location` + `range.length`.

Availability
Available in iOS 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 iOS 2.0 and later.

See Also
`NSHomeDirectory` (page 1704)
`NSHomeDirectoryForUser` (page 1704)

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 iOS 2.0 and later.

See Also
`NSRoundDownToMultipleOfPageSize` (page 1715)
`NSRoundUpToMultipleOfPageSize` (page 1715)
`NSLogPageSize` (page 1709)

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.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 1709)
`NSLogv` (page 1709)
`NSAssert` (page 1679)
`NSCAssert` (page 1685)
`NSCParameterAssert` (page 1693)

Declared In
`NSException.h`

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 iOS 2.0 and later.

See Also
`NSStringFromProtocol` (page 1719)
`NSClassFromString` (page 1689)
`NSSelectorFromString` (page 1716)

Declared In
`NSObjCRuntime.h`

NSRangeFromString

Returns a range from a textual 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 iOS 2.0 and later.

See Also
`NSStringFromRange` (page 1719)

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 iOS 2.0 and later.

Declared In
`NSZone.h`

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.)

Availability
Available in iOS 2.0 and later.

See Also
`NSCreateZone` (page 1693)
`NSZoneMalloc` (page 1735)

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 iOS 2.0 and later.

See Also
NSPageSize (page 1712)
NSLogPageSize (page 1709)
NSRoundUpToMultipleOfPageSize (page 1715)

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 iOS 2.0 and later.

See Also
NSPageSize (page 1712)
NSLogPageSize (page 1709)
NSRoundDownToMultipleOfPageSize (page 1715)

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 1267).

For more information on file system utilities, see Locating Directories on the System.

Note: The directory returned by this method may not exist. This method simply gives you the appropriate location for the requested directory. Depending on the application's needs, it may be up to the developer to create the appropriate directory and any in between.

Availability
Available in iOS 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 Objects, Classes, and Messaging in *The Objective-C Programming Language*.

Availability
Available in iOS 2.0 and later.

See Also
NSStringFromSelector (page 1719)
NSProtocolFromString (page 1713)
NSClassFromString (page 1689)

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 iOS 2.0 and later.

See Also
NSGetUncaughtExceptionHandler (page 1704)

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 iOS 2.0 and later.

See Also
NSZoneName (page 1736)

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 954), when deciding whether to retain *anObject* as opposed to making a copy of it.

Availability

Available in iOS 2.0 and later.

Declared In

`NSObject.h`

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 iOS 2.0 and later.

See Also

`NSClassFromString` (page 1689)

`NSStringFromProtocol` (page 1719)

`NSStringFromSelector` (page 1719)

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 iOS 2.0 and later.

See Also

NSProtocolFromString (page 1713)
NSStringFromClass (page 1718)
NSStringFromSelector (page 1719)

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 iOS 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 iOS 2.0 and later.

See Also
NSSelectorFromString (page 1716)
NSStringFromProtocol (page 1719)
NSStringFromClass (page 1718)

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 1722) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostDoubleToBig (page 1723)
NSSwapLittleDoubleToHost (page 1729)

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 1723) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostFloatToBig (page 1724)

NSSwapLittleFloatToHost (page 1729)

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 1728) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostIntToBig (page 1725)
NSSwapLittleIntToHost (page 1730)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostLongLongToBig (page 1726)
NSSwapLittleLongLongToHost (page 1730)

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 1731) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapHostLongToBig` (page 1727)
`NSSwapLittleLongToHost` (page 1730)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapHostShortToBig` (page 1727)
`NSSwapLittleShortToHost` (page 1731)

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 iOS 2.0 and later.

See Also
NSSwapLongLong (page 1732)
NSSwapFloat (page 1723)

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 iOS 2.0 and later.

See Also
NSSwapLong (page 1731)
NSSwapDouble (page 1722)

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 1722) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapBigDoubleToHost (page 1720)
NSSwapHostDoubleToLittle (page 1724)

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 1722) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapLittleDoubleToHost` (page 1729)
`NSSwapHostDoubleToBig` (page 1723)

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 1723) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapBigFloatToHost` (page 1720)
`NSSwapHostFloatToLittle` (page 1724)

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 1723) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapLittleFloatToHost` (page 1729)
`NSSwapHostFloatToBig` (page 1724)

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 1728) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapBigIntToHost` (page 1721)
`NSSwapHostIntToLittle` (page 1725)

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 1728) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapLittleIntToHost (page 1730)
NSSwapHostIntToBig (page 1725)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapBigLongLongToHost (page 1721)
NSSwapHostLongLongToLittle (page 1726)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapLittleLongLongToHost (page 1730)
NSSwapHostLongLongToBig (page 1726)

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 1731) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapBigLongToHost` (page 1721)
`NSSwapHostLongToLittle` (page 1727)

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 1731) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapLittleLongToHost` (page 1730)
`NSSwapHostLongToBig` (page 1727)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapBigShortToHost` (page 1722)
`NSSwapHostShortToLittle` (page 1728)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapLittleShortToHost` (page 1731)
`NSSwapHostShortToBig` (page 1727)

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 iOS 2.0 and later.

See Also
NSSwapShort (page 1732)
NSSwapLong (page 1731)
NSSwapLongLong (page 1732)

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 1722) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostDoubleToLittle (page 1724)
NSSwapBigDoubleToHost (page 1720)
NSConvertSwappedDoubleToHost (page 1691)

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 1723) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostFloatToLittle (page 1724)
NSSwapBigFloatToHost (page 1720)

NSConvertSwappedFloatToHost (page 1691)

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 1728) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostIntToLittle (page 1725)
NSSwapBigIntToHost (page 1721)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostLongLongToLittle (page 1726)
NSSwapBigLongLongToHost (page 1721)

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 1731) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapHostLongToLittle` (page 1727)
`NSSwapBigLongToHost` (page 1721)
`NSSwapLong` (page 1731)

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 1732) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapHostShortToLittle` (page 1728)
`NSSwapBigShortToHost` (page 1722)

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 iOS 2.0 and later.

See Also
NSSwapLongLong (page 1732)
NSSwapInt (page 1728)
NSSwapFloat (page 1723)

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 iOS 2.0 and later.

See Also
NSSwapLong (page 1731)
NSSwapDouble (page 1722)

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 iOS 2.0 and later.

See Also
NSSwapInt (page 1728)
NSSwapLong (page 1731)

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*.

The temporary directory is determined by `confstr`(3) passing the `_CS_DARWIN_USER_TEMP_DIR` flag. The erase rules are whatever match that directory.

See the `NSFileManager` method `URLForDirectory:inDomain:appropriateForURL:create:error:` (page 534) for an alternate (and more flexible) means of finding the correct temporary directory.

Availability
Available in iOS 2.0 and later.

See Also
`NSSearchPathForDirectoriesInDomains` (page 1715)
`NSHomeDirectory` (page 1704)

Related Sample Code
SpeakHere

Declared In
NSPathUtilities.h

NSUnionRange

Returns the union 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 iOS 2.0 and later.

See Also
`NSIntersectionRange` (page 1706)

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 iOS 2.0 and later.

See Also
NSFullUserName (page 1703)
NSHomeDirectory (page 1704)
NSHomeDirectoryForUser (page 1704)

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 iOS 2.0 and later.

See Also
NSDefaultMallocZone (page 1702)
NSRecycleZone (page 1714)
NSZoneFree (page 1735)
NSZoneMalloc (page 1735)
NSZoneRealloc (page 1736)

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 iOS 2.0 and later.

See Also
`NSRecycleZone` (page 1714)
`NSZoneMalloc` (page 1735)
`NSZoneCalloc` (page 1734)
`NSZoneRealloc` (page 1736)

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 iOS 2.0 and later.

See Also
`NSZoneCalloc` (page 1734)
`NSZoneMalloc` (page 1735)
`NSZoneRealloc` (page 1736)

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 iOS 2.0 and later.

See Also
`NSDefaultMallocZone` (page 1702)
`NSRecycleZone` (page 1714)
`NSZoneFree` (page 1735)
`NSZoneCalloc` (page 1734)
`NSZoneRealloc` (page 1736)

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 iOS 2.0 and later.

See Also
`NSSetZoneName` (page 1717)

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 iOS 2.0 and later.

See Also
`NSDefaultMallocZone` (page 1702)
`NSRecycleZone` (page 1714)
`NSZoneFree` (page 1735)
`NSZoneCalloc` (page 1734)
`NSZoneMalloc` (page 1735)

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 1738)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.

Availability
Available in iOS 2.0 and later.

Declared In
`NSException.h`

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 1738)macros marks the beginning of this section.) If an exception is raised in the exception handling domain marked off by the `NS_DURING` (page 1737) and `NS_HANDLER` (page 1738), the local exception handler (if specified) is given a chance to handle the exception.

Availability
Available in iOS 2.0 and later.

Declared In
`NSException.h`

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 1737) macro marks the beginning of the exception-handling domain; the `NS_ENDHANDLER` (page 1737) 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.

Availability
Available in iOS 2.0 and later.

Declared In
`NSException.h`

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 1737) and `NS_HANDLER` (page 1738) 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.

Availability
Available in iOS 2.0 and later.

Declared In
`NSException.h`

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 1737) and `NS_HANDLER` (page 1738) macros that might raise an exception. The standard `return` statement does not work as expected in the exception-handling domain.

Availability
Available in iOS 2.0 and later.

Declared In
`NSException.h`

Data Types

PART IV

Data Types

Foundation Data Types Reference

Framework: Foundation/Foundation.h

Overview

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 iOS 2.0 and later.
Declared in `NSByteOrder.h`.

`NS_LittleEndian`
The byte order is little endian.
Available in iOS 2.0 and later.
Declared in `NSByteOrder.h`.

`NS_BigEndian`
The byte order is big endian.
Available in iOS 2.0 and later.
Declared in `NSByteOrder.h`.

Discussion

These constants are returned by `NSHostByteOrder` (page 1705).

NSComparator

Defines the signature for a block object used for comparison operations.

```
typedef NSComparisonResult (^NSComparator)(id obj1, id obj2);
```

Discussion

The arguments to the block are two objects to compare. The block returns an `NSComparisonResult` (page 1744) value to denote the ordering of the two objects.

You use `NSComparator` blocks in comparison operations such as `NSArray`'s `sortedArrayUsingComparator:` (page 77), for example:

```
NSArray *sortedArray = [array sortedArrayUsingComparator: ^(id obj1, id obj2)
{

    if ([obj1 integerValue] > [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedDescending;
    }

    if ([obj1 integerValue] < [obj2 integerValue]) {
        return (NSComparisonResult)NSOrderedAscending;
    }
    return (NSComparisonResult)NSOrderedSame;
}];
```

Availability

Available in iOS 4.0 and later.

Declared In

`NSObjCRuntime.h`

NSComparisonResult

These constants are used to indicate how items in a request are ordered.

```
enum {
    NSOrderedAscending = -1,
    NSOrderedSame,
    NSOrderedDescending
};
typedef NSInteger NSComparisonResult;
```

Constants

`NSOrderedAscending`

The left operand is smaller than the right operand.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSOrderedSame`

The two operands are equal.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSOrderedDescending`
: The left operand is greater than the right operand.

  Available in iOS 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 iOS 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 1674).

Availability
Available in iOS 2.0 and later.

Declared In
`NSDecimal.h`

NSEnumerationOptions

Type to specify behavior during enumeration.

```
typedef NSUInteger NSEnumerationOptions;
```

Availability
Available in iOS 4.0 and later.

Declared In
`NSObjCRuntime.h`

NSInteger

Used to describe an integer.

```
#if __LP64__ || TARGET_OS_EMBEDDED || TARGET_OS_IPHONE || TARGET_OS_WIN32 ||
NS_BUILD_32_LIKE_64
typedef long NSInteger;
#else
typedef int NSInteger;
#endif
```

Discussion

When building 32-bit applications, NSInteger is a 32-bit integer. A 64-bit application treats NSInteger as a 64-bit integer.

Availability

Available in iOS 2.0 and later.

Declared In

`NSObjCRuntime.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 1702)
- `NSIntersectionRange` (page 1706)
- `NSLocationInRange` (page 1708)
- `NSMakeRange` (page 1711)
- `NSMaxRange` (page 1711)
- `NSRangeFromString` (page 1713)
- `NSStringFromRange` (page 1719)
- `NSUnionRange` (page 1733)

Availability

Available in iOS 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 iOS 2.0 and later.

Declared In
`NSRange.h`

NSSearchPathDirectory

These constants specify the location of a variety of directories.

```
enum {
    NSApplicationDirectory = 1,
    NSDemoApplicationDirectory,
    NSDeveloperApplicationDirectory,
    NSAdminApplicationDirectory,
    NSLibraryDirectory,
    NSDeveloperDirectory,
    NSUserDirectory,
    NSDocumentationDirectory,
    NSDocumentDirectory,
    NSCoreServiceDirectory,
    NSAutosavedInformationDirectory = 11,
    NSDesktopDirectory = 12,
    NSCachesDirectory = 13,
    NSApplicationSupportDirectory = 14,
    NSDownloadsDirectory = 15,
    NSInputMethodsDirectory = 16,
    NSMoviesDirectory = 17,
    NSMusicDirectory = 18,
    NSPicturesDirectory = 19,
    NSPrinterDescriptionDirectory = 20,
    NSSharedPublicDirectory = 21,
    NSPreferencePanesDirectory = 22,
    NSItemReplacementDirectory = 99,
    NSAllApplicationsDirectory = 100,
    NSAllLibrariesDirectory = 101
};
typedef NSUInteger NSSearchPathDirectory;
```

Constants

`NSApplicationDirectory`
Supported applications (`/Applications`).
Available in iOS 2.0 and later.
Declared in `NSPathUtilities.h`.

`NSDemoApplicationDirectory`
Unsupported applications and demonstration versions.
Available in iOS 2.0 and later.
Declared in `NSPathUtilities.h`.

`NSDeveloperApplicationDirectory`
: Developer applications (`/Developer/Applications`).

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSAdminApplicationDirectory`
: System and network administration applications.

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSLibraryDirectory`
: Various user-visible documentation, support, and configuration files (`/Library`).

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSDeveloperDirectory`
: Developer resources (`/Developer`).

  Deprecated: Beginning with Xcode 3.0, developer tools can be installed in any location.

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSUserDirectory`
: User home directories (`/Users`).

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSDocumentationDirectory`
: Documentation.

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSDocumentDirectory`
: Document directory.

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSCoreServiceDirectory`
: Location of core services (`System/Library/CoreServices`).

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSAutosavedInformationDirectory`
: Location of user's autosaved documents `Library/Autosave Information`

  Available in iOS 4.0 and later.

  Declared in `NSPathUtilities.h`.

`NSDesktopDirectory`
: Location of user's desktop directory.

  Available in iOS 2.0 and later.

  Declared in `NSPathUtilities.h`.

`NSCachesDirectory`
Location of discardable cache files (`Library/Caches`).

Available in iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

`NSApplicationSupportDirectory`
Location of application support files (`Library/Application Support`).

Available in iOS 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 iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

`NSInputMethodsDirectory`
Location of Input Methods (*Library/Input Methods*)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSMoviesDirectory`
Location of user's Movies directory (`~/Movies`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSMusicDirectory`
Location of user's Movies directory (`~/Music`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSPicturesDirectory`
Location of user's Movies directory (`~/Pictures`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSPrinterDescriptionDirectory`
Location of system's PPDs directory (`Library/Printers/PPDs`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSSharedPublicDirectory`
Location of user's Public sharing directory (`~/Public`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSPreferencePanesDirectory`
Location of the PreferencePanes directory for use with System Preferences (`Library/PreferencePanes`)

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSItemReplacementDirectory`
For use with `NSFileManager` method `URLForDirectory:inDomain:appropriateForURL:create:error:`

Available in iOS 4.0 and later.

Declared in `NSPathUtilities.h`.

`NSAllApplicationsDirectory`
All directories where applications can occur.

Available in iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

`NSAllLibrariesDirectory`
All directories where resources can occur.

Available in iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSPathUtilities.h`

NSSearchPathDomainMask

Search path domain constants specifying base locations for the NSSearchPathDirectory (page 1747) type.

```
enum {
    NSUserDomainMask = 1,
    NSLocalDomainMask = 2,
    NSNetworkDomainMask = 4,
    NSSystemDomainMask = 8,
    NSAllDomainsMask = 0xffff,
};
typedef NSUInteger NSSearchPathDomainMask;
```

Constants
`NSUserDomainMask`
The user's home directory—the place to install user's personal items (~).

Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

`NSSystemDomainMask`
: Provided by Apple — can't be modified (`/System`) .

Available in iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

`NSAllDomainsMask`
: All domains.

Includes all of the above and future items.

Available in iOS 2.0 and later.

Declared in `NSPathUtilities.h`.

Availability
Available in iOS 2.0 and later.

Declared In
`NSPathUtilities.h`

NSSocketNativeHandle

Type for the platform-specific native socket handle.

```
typedef int NSSocketNativeHandle;
```

Availability
Available in iOS 2.0 and later.

Declared In
`NSPort.h`

NSSortOptions

Type to specify behavior during sort operations.

```
typedef NSUInteger NSSortOptions;
```

Availability
Available in iOS 4.0 and later.

Declared In
`NSObjCRuntime.h`

NSStringEncoding

Type representing string-encoding values.

```
typedef NSUInteger NSStringEncoding;
```

Discussion
See String Encodings (page 1283) for a list of values.

Availability
Available in iOS 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 iOS 2.0 and later.

Declared In
NSByteOrder.h

NSSwappedFloat

Opaque type containing an endian-independent `float` value.

```
typedef struct {
    unsigned int v;
} NSSwappedFloat;
```

Discussion
The fields of an `NSSwappedFloat` are private.

Availability
Available in iOS 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 iOS 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 1717).

Declared In
`NSException.h`

NSUInteger

Used to describe an unsigned integer.

```
#if __LP64__ || TARGET_OS_EMBEDDED || TARGET_OS_IPHONE || TARGET_OS_WIN32 ||
NS_BUILD_32_LIKE_64
typedef unsigned long NSUInteger;
#else
typedef unsigned int NSUInteger;
#endif
```

Discussion
When building 32-bit applications, NSUInteger is a 32-bit unsigned integer. A 64-bit application treats NSUInteger as a 64-bit unsigned integer

Availability
Available in iOS 2.0 and later.

Declared In
`NSObjCRuntime.h`

NSZone

Used to identify and manage memory zones.

```
typedef struct _NSZone NSZone;
```

Availability
Available in iOS 2.0 and later.

Declared In
`NSZone.h`

Constants

Foundation Constants Reference

Framework: Foundation/Foundation.h

Overview

This document defines constants in the Foundation framework that are not associated with a particular class.

Constants

Enumerations

NSNotFound

Defines a value that indicates that an item requested couldn't be found or doesn't exist.

```
enum {
   NSNotFound = NSIntegerMax
};
```

Constants

`NSNotFound`

A value that indicates that an item requested couldn't be found or doesn't exist.

Available in iOS 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.

Special Considerations

Prior to Mac OS X v10.5, `NSNotFound` was defined as `0x7fffffff`. For 32-bit systems, this was effectively the same as `NSIntegerMax`. To support 64-bit environments, `NSNotFound` is now formally defined as `NSIntegerMax`. This means, however, that the value is different in 32-bit and 64-bit environments. You should therefore not save the value directly in files or archives. Moreover, sending the value between 32-bit and 64-bit processes via Distributed Objects will not get you `NSNotFound` on the other side. This applies to any Cocoa methods invoked over Distributed Objects and which might return `NSNotFound`, such as the `indexOfObject:` method of `NSArray` (if sent to a proxy for an array).

Enumeration Options

Options for Block enumeration operations.

```
enum {
    NSEnumerationConcurrent = (1UL << 0),
    NSEnumerationReverse = (1UL << 1),
};
```

Constants

`NSEnumerationConcurrent`

Specifies that the Block enumeration should be concurrent.

The order of invocation is nondeterministic and undefined; this flag is a hint and may be ignored by the implementation under some circumstances; the code of the Block must be safe against concurrent invocation.

Available in iOS 4.0 and later.

Declared in `NSObjCRuntime.h`.

`NSEnumerationReverse`

Specifies that the enumeration should be performed in reverse.

This option is available for `NSArray` and `NSIndexSet` classes; its behavior is undefined for `NSDictionary` and `NSSet` classes, or when combined with the `NSEnumerationConcurrent` flag.

Available in iOS 4.0 and later.

Declared in `NSObjCRuntime.h`.

Declared In

`NSObjCRuntime.h`

Sort Options

Options for Block sorting operations.

```
enum {
    NSSortConcurrent = (1UL << 0),
    NSSortStable = (1UL << 4),
};
```

Constants

`NSSortConcurrent`

Specifies that the Block sort operation should be concurrent.

This option is a hint and may be ignored by the implementation under some circumstances; the code of the Block must be safe against concurrent invocation.

Available in iOS 4.0 and later.

Declared in `NSObjCRuntime.h`.

`NSSortStable`

Specifies that the sorted results should return compared items have equal value in the order they occurred originally.

If this option is unspecified equal objects may, or may not, be returned in their original order.

Available in iOS 4.0 and later.

Declared in `NSObjCRuntime.h`.

Declared In
NSObjCRuntime.h

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,
    NSFileReadTooLargeError = 263,
    NSFileReadUnknownStringEncodingError = 264,
    NSFileWriteUnknownError = 512,
    NSFileWriteNoPermissionError = 513,
    NSFileWriteInvalidFileNameError = 514,
    NSFileWriteInapplicableStringEncodingError = 517,
    NSFileWriteUnsupportedSchemeError = 518,
    NSFileWriteOutOfSpaceError = 640,
    NSFileWriteVolumeReadOnlyError = 642m
    NSKeyValueValidationError = 1024,
    NSFormattingError = 2048,
    NSUserCancelledError = 3072,

    NSFileErrorMinimum = 0,
    NSFileErrorMaximum = 1023,
    NSValidationErrorMinimum = 1024,
    NSValidationErrorMaximum = 2047,
    NSFormattingErrorMinimum = 2048,
    NSFormattingErrorMaximum = 2559,

    NSPropertyListReadCorruptError = 3840,
    NSPropertyListReadUnknownVersionError = 3841,
    NSPropertyListReadStreamError = 3842,
    NSPropertyListWriteStreamError = 3851,
    NSPropertyListErrorMinimum = 3840,
    NSPropertyListErrorMaximum = 4095

    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 iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileLockingError`
Failure to get a lock on file

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadUnknownError`
Read error, reason unknown

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadNoPermissionError`
Read error because of a permission problem

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadInvalidFileNameError`
Read error because of an invalid file name

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadCorruptFileError`
Read error because of a corrupted file, bad format, or similar reason

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadNoSuchFileError`
Read error because no such file was found

Available in iOS 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 iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadUnsupportedSchemeError`
Read error because the specified URL scheme is unsupported

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadTooLargeError`
Read error because the specified file was too large.

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileReadUnknownStringEncodingError`
Read error because the string coding of the file could not be determined

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteUnknownError`
Write error, reason unknown

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteNoPermissionError`
Write error because of a permission problem

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteInvalidFileNameError`
Write error because of an invalid file name

Available in iOS 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 iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteUnsupportedSchemeError`
Write error because the specified URL scheme is unsupported

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteOutOfSpaceError`
Write error because of a lack of disk space

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFileWriteVolumeReadOnlyError`
Write error because because the volume is read only.

Available in iOS 4.0 and later.

Declared in `FoundationErrors.h`.

`NSKeyValueValidationError`
Key-value coding validation error

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSFormattingError`
Formatting error (related to display of data)
Available in iOS 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 iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSFileErrorMinimum`
Marks the start of the range of error codes reserved for file errors
Available in iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSFileErrorMaximum`
Marks the end of the range of error codes reserved for file errors
Available in iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSValidationErrorMinimum`
Marks the start of the range of error codes reserved for validation errors.
Available in iOS 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 iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSFormattingErrorMinimum`
Marks the start of the range of error codes reserved for formatting errors.
Available in iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSFormattingErrorMaximum`
Marks end of the range of error codes reserved for formatting errors.
Available in iOS 2.0 and later.
Declared in `FoundationErrors.h`.

`NSPropertyListReadCorruptError`
An error was encountered while parsing the property list.
Available in iOS 4.0 and later.
Declared in `FoundationErrors.h`.

`NSPropertyListReadUnknownVersionError`
The version number of the property list is unable to be determined.
Available in iOS 4.0 and later.
Declared in `FoundationErrors.h`.

`NSPropertyListReadStreamError`
An stream error was encountered while reading the property list.

Available in iOS 4.0 and later.

Declared in `FoundationErrors.h`.

`NSPropertyListWriteStreamError`
An stream error was encountered while writing the property list.

Available in iOS 4.0 and later.

Declared in `FoundationErrors.h`.

`NSPropertyListErrorMinimum`
Marks beginning of the range of error codes reserved for property list errors.

Available in iOS 4.0 and later.

Declared in `FoundationErrors.h`.

`NSPropertyListErrorMaximum`
Marks end of the range of error codes reserved for property list errors.

Available in iOS 4.0 and later.

Declared in `FoundationErrors.h`.

`NSExecutableErrorMinimum`
Marks beginning of the range of error codes reserved for errors related to executable files.

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSExecutableNotLoadableError`
Executable is of a type that is not loadable in the current process.

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSExecutableArchitectureMismatchError`
Executable does not provide an architecture compatible with the current process.

Available in iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSExecutableRuntimeMismatchError`
Executable has Objective C runtime information incompatible with the current process.

Available in iOS 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 iOS 2.0 and later.

Declared in `FoundationErrors.h`.

`NSExecutableLinkError`
Executable fails due to linking issues.

Available in iOS 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 iOS 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`

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,
    NSURLErrorCannotDecodeRawData = -1015,
    NSURLErrorCannotDecodeContentData = -1016,
    NSURLErrorCannotParseResponse = -1017,
    NSURLErrorInternationalRoamingOff = -1018,
    NSURLErrorCallIsActive = -1019,
    NSURLErrorDataNotAllowed = -1020,
    NSURLErrorRequestBodyStreamExhausted = -1021,
    NSURLErrorFileDoesNotExist = -1100,
    NSURLErrorFileIsDirectory = -1101,
    NSURLErrorNoPermissionsToReadFile = -1102,
    NSURLErrorSecureConnectionFailed = -1200,
    NSURLErrorServerCertificateHasBadDate = -1201,
    NSURLErrorServerCertificateUntrusted = -1202,
    NSURLErrorServerCertificateHasUnknownRoot = -1203,
    NSURLErrorServerCertificateNotYetValid = -1204,
    NSURLErrorClientCertificateRejected = -1205,
    NSURLErrorClientCertificateRequired = -1206,
    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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotFindHost`
: Returned when the host name for a URL cannot be resolved.

  Available in iOS 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 iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDataLengthExceedsMaximum`
: Returned when the length of the resource data exceeds the maximum allowed.

  Available in iOS 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 iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDNSLookupFailed`
See `NSURLErrorCannotFindHost`

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorUserAuthenticationRequired`
Returned when authentication is required to access a resource.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorCannotDecodeRawData`
Returned when content data received during an NSURLConnection request cannot be decoded for a known content encoding.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorCannotDecodeContentData`
Returned when content data received during an NSURLConnection request has an unknown content encoding.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorCannotParseResponse`
Returned when a response to an NSURLConnection request cannot be parsed.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorInternationalRoamingOff`
Returned when a connection would require activating a data context while roaming, but international roaming is disabled.

Available in iOS 3.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorCallIsActive`
Returned when a connection is attempted while a phone call is active on a network that does not support simultaneous phone and data communication (EDGE or GPRS).

Available in iOS 3.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorDataNotAllowed`
Returned when the cellular network disallows a connection.

Available in iOS 3.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorRequestBodyStreamExhausted`
Returned when a body stream is needed but the client does not provide one. This impacts clients on iOS that send a POST request using a body stream but do not implement the NSURLConnection delegate method `connection:needNewBodyStream`.

Available in iOS 3.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorFileDoesNotExist`
Returned when a file does not exist.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorNoPermissionsToReadFile`
Returned when a resource cannot be read due to insufficient permissions.

Available in iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorServerCertificateHasUnknownRoot`
Returned when a server certificate is not signed by any root server.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorServerCertificateNotYetValid`
Returned when a server certificate is not yet valid.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorClientCertificateRejected`
Returned when a server certificate is rejected.

Available in iOS 2.0 and later.

Declared in `NSURLError.h`.

`NSURLErrorClientCertificateRequired`
Returned when a client certificate is required to authenticate an SSL connection during an NSURLConnection request.

Available in iOS 4.0 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 iOS 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 iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotOpenFile`
: Returned when `NSURLDownload` was unable to open the downloaded file on disk.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotCloseFile`
: Returned when `NSURLDownload` was unable to close the downloaded file on disk.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotWriteToFile`
: Returned when `NSURLDownload` was unable to write to the downloaded file on disk.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotRemoveFile`
: Returned when `NSURLDownload` was unable to remove a downloaded file from disk.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorCannotMoveFile`
: Returned when `NSURLDownload` was unable to move a downloaded file on disk.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDownloadDecodingFailedMidStream`
: Returned when `NSURLDownload` failed to decode an encoded file during the download.

  Available in iOS 2.0 and later.

  Declared in `NSURLError.h`.

`NSURLErrorDownloadDecodingFailedToComplete`
: Returned when `NSURLDownload` failed to decode an encoded file after downloading.

  Available in iOS 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 iOS 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 iOS 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 1745).

Gives a precision of at least 38 decimal digits, 128 binary positions.

Available in iOS 2.0 and later.

Declared in `NSDecimal.h`.

`NSDecimalNoScale`

Specifies that the number of digits allowed after the decimal separator in a decimal number should not be limited.

Available in iOS 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 iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSIntegerMin`

The minimum value for an `NSInteger`.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSUIntegerMax`

The maximum value for an `NSUInteger`.

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSException.h`.

`NSMallocException`

Obsolete; not currently used.

Available in iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 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 iOS 2.0 and later.

  Declared in `NSException.h`.

`NSPortSendException`
: Generic error occurred on send.

  This is an `NSPort`-specific exception.

  Available in iOS 2.0 and later.

  Declared in `NSException.h`.

`NSPortReceiveException`
: Generic error occurred on receive.

  This is an `NSPort`-specific exception.

  Available in iOS 2.0 and later.

  Declared in `NSException.h`.

`NSOldStyleException`
No longer used.

Available in iOS 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 iOS 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
#define NSFoundationVersionNumber10_4_8    567.28
#define NSFoundationVersionNumber10_4_9    567.29
#define NSFoundationVersionNumber10_4_10    567.29
#define NSFoundationVersionNumber10_4_11    567.36
#define NSFoundationVersionNumber10_5    677.00
#define NSFoundationVersionNumber10_5_1    677.10
#define NSFoundationVersionNumber10_5_2 677.15
#define NSFoundationVersionNumber10_5_3 677.19
#define NSFoundationVersionNumber10_5_4 677.19
#define NSFoundationVersionNumber10_5_5 677.21
#define NSFoundationVersionNumber10_5_6 677.22
#define NSFoundationVersionNumber_iOS_2_0    678.24
#define NSFoundationVersionNumber_iOS_2_1  678.26
#define NSFoundationVersionNumber_iOS_2_2  678.29
```

Constants

`NSFoundationVersionNumber10_0`

Foundation version released in Mac OS X version 10.0.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1`
Foundation version released in Mac OS X version 10.1.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_1`
Foundation version released in Mac OS X version 10.1.1.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_2`
Foundation version released in Mac OS X version 10.1.2.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_3`
Foundation version released in Mac OS X version 10.1.3.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_1_4`
Foundation version released in Mac OS X version 10.1.4.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2`
Foundation version released in Mac OS X version 10.2.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_1`
Foundation version released in Mac OS X version 10.2.1.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_2`
Foundation version released in Mac OS X version 10.2.2.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_3`
Foundation version released in Mac OS X version 10.2.3.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_4`
Foundation version released in Mac OS X version 10.2.4.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_5`
Foundation version released in Mac OS X version 10.2.5.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_6`
Foundation version released in Mac OS X version 10.2.6.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_7`
Foundation version released in Mac OS X version 10.2.7.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_2_8`
Foundation version released in Mac OS X version 10.2.8.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3`
Foundation version released in Mac OS X version 10.3.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_1`
Foundation version released in Mac OS X version 10.3.1.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_2`
Foundation version released in Mac OS X version 10.3.2.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_3`
Foundation version released in Mac OS X version 10.3.3.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_4`
Foundation version released in Mac OS X version 10.3.4.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_5`
Foundation version released in Mac OS X version 10.3.5.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_6`
Foundation version released in Mac OS X version 10.3.6.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_7`
Foundation version released in Mac OS X version 10.3.7.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_8`
Foundation version released in Mac OS X version 10.3.8.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_3_9`
Foundation version released in Mac OS X version 10.3.9.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4`
Foundation version released in Mac OS X version 10.4.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_1`
Foundation version released in Mac OS X version 10.4.1.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_2`
Foundation version released in Mac OS X version 10.4.2.

Available in iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_3`
Foundation version released in Mac OS X version 10.4.3.

Available in iOS 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 iOS 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 iOS 2.0 and later.

Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_5`
Foundation version released in Mac OS X version 10.4.5.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_6`
Foundation version released in Mac OS X version 10.4.6.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_7`
Foundation version released in Mac OS X version 10.4.7.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_8`
Foundation version released in Mac OS X version 10.4.8.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_9`
Foundation version released in Mac OS X version 10.4.9.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_10`
Foundation version released in Mac OS X version 10.4.10.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_4_11`
Foundation version released in Mac OS X version 10.4.11.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5`
Foundation version released in Mac OS X version 10.5.0.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_1`
Foundation version released in Mac OS X version 10.5.1.
Available in iOS 2.0 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_2`
Foundation version released in Mac OS X version 10.5.2.
Available in iOS 2.2 and later.
Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_3`
: Foundation version released in Mac OS X version 10.5.3.

  Available in iOS 2.2 and later.

  Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_4`
: Foundation version released in Mac OS X version 10.5.4.

  Available in iOS 2.2 and later.

  Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_5`
: Foundation version released in Mac OS X version 10.5.5.

  Available in iOS 4.0 and later.

  Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber10_5_6`
: Foundation version released in Mac OS X version 10.5.6.

  Available in iOS 4.0 and later.

  Declared in `NSObjCRuntime.h`.

`NSFoundationVersionNumber_iOS_2_0`
: Foundation version released in iOS version 2.0.

  This constant is only available on iOS.

`NSFoundationVersionNumber_iOS_2_1`
: Foundation version released in iOS version 2.1.

  This constant is only available on iOS.

`NSFoundationVersionNumber_iOS_2_2`
: Foundation version released in iOS version 2.2.

  This constant is only available on iOS.

Declared In
`NSObjCRuntime.h`

Document Revision History

This table describes the changes to *Foundation Framework Reference*.

Date	Notes
2010-05-20	Updated the list of classes and protocols for iOS 4.0.
2009-08-28	Updated class hierarchy diagrams.
2009-05-19	Added new classes for Mac OS X v10.6.
2008-06-27	Updated for iOS.
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