NSArray Class Reference

Data Management: Data Types & Collections

2010-04-29

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

Simultaneously published in the United States and Canada.

Contents

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	NSArray.h NSKeyValueCoding.h NSKeyValueObserving.h NSPathUtilities.h NSPredicate.h NSSortDescriptor.h
Companion guides	Collections Programming Topics Key-Value Coding Programming Guide Property List Programming Guide Predicate Programming Guide
Related sample code	GKRocket ScrollViewSuite SpeakHere ToolbarSearch WiTap

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 22) and objectAtIndex: (page 42)—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 43) and reverseObjectEnumerator (page 46) 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 31) 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: message, as declared in the NSObject protocol. Another method, indexOfObjectIdenticalTo: (page 35), 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 26) method allows you to create a new array from an existing array filtered using a predicate (see *Predicate Programming Guide*).

NSArray's makeObjectsPerformSelector: (page 41) and makeObjectsPerformSelector:withObject: (page 42) 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 49) and sortedArrayUsingFunction:context: (page 48), extract a subset of the array (subarrayWithRange: (page 51)), or concatenate the elements of an array of NSString objects into a single string (componentsJoinedByString: (page 21)). In addition, you can compare two arrays using the isEqualToArray: (page 41) and firstObjectCommonWithArray: (page 27) methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with arrayByAddingObject: (page 20) and arrayByAddingObjectsFromArray: (page 20).

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 22) and `objectAtIndex:` (page 42). 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` 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` to each object that you add to your collection and `release` 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:`
- `initWithCoder:`

NSCopying

`copyWithZone:`

NSMutableCopying
`mutableCopyWithZone:`

Tasks

Creating an Array

`+ array` (page 14)
Creates and returns an empty array.

`+ arrayWithArray:` (page 15)
Creates and returns an array containing the objects in another given array.

`+ arrayWithContentsOfFile:` (page 15)
Creates and returns an array containing the contents of the file specified by a given path.

`+ arrayWithContentsOfURL:` (page 16)
Creates and returns an array containing the contents specified by a given URL.

`+ arrayWithObject:` (page 16)
Creates and returns an array containing a given object.

`+ arrayWithObjects:` (page 17)
Creates and returns an array containing the objects in the argument list.

`+ arrayWithObjects:count:` (page 18)
Creates and returns an array that includes a given number of objects from a given C array.

Initializing an Array

`- initWithArray:` (page 37)
Initializes a newly allocated array by placing in it the objects contained in a given array.

`- initWithArray:copyItems:` (page 38)
Initializes a newly allocated array using *`anArray`* as the source of data objects for the array.

`- initWithContentsOfFile:` (page 38)
Initializes a newly allocated array with the contents of the file specified by a given path.

`- initWithContentsOfURL:` (page 39)
Initializes a newly allocated array with the contents of the location specified by a given URL.

`- initWithObjects:` (page 39)
Initializes a newly allocated array by placing in it the objects in the argument list.

`- initWithObjects:count:` (page 40)
Initializes a newly allocated array to include a given number of objects from a given C array.

Querying an Array

- `containsObject:` (page 21)
  Returns a Boolean value that indicates whether a given object is present in the receiver.
- `count` (page 22)
  Returns the number of objects currently in the receiver.
- `getObjects:range:` (page 27)
  Copies the objects contained in the receiver that fall within the specified range to *aBuffer*.
- `lastObject` (page 41)
  Returns the object in the array with the highest index value.
- `objectAtIndex:` (page 42)
  Returns the object located at *index*.
- `objectsAtIndexes:` (page 44)
  Returns an array containing the objects in the receiver at the indexes specified by a given index set.
- `objectEnumerator` (page 43)
  Returns an enumerator object that lets you access each object in the receiver.
- `reverseObjectEnumerator` (page 46)
  Returns an enumerator object that lets you access each object in the receiver, in reverse order.
- `getObjects:` (page 55) Deprecated in iOS 4.0
  Copies all the objects contained in the receiver to *aBuffer*.

Finding Objects in an Array

- `indexOfObject:` (page 31)
  Returns the lowest index whose corresponding array value is equal to a given object.
- `indexOfObject:inRange:` (page 32)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectIdenticalTo:` (page 35)
  Returns the lowest index whose corresponding array value is identical to a given object.
- `indexOfObjectIdenticalTo:inRange:` (page 35)
  Returns the lowest index within a specified range whose corresponding array value is equal to a given object .
- `indexOfObjectPassingTest:` (page 36)
  Returns the index of the first object in the receiver that passes a test in a given Block.
- `indexOfObjectWithOptions:passingTest:` (page 36)
  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 34)
  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 29)
  Returns the indexes of objects in the receiver that pass a test in a given Block.

- `indexesOfObjectsWithOptions:passingTest:` (page 30)
  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 28)
  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 32)
  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 41)
  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 42)
  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 25)
  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 26)
  Executes a given block using each object in the receiver.
- `enumerateObjectsAtIndexes:options:usingBlock:` (page 24)
  Executes a given block using the objects in the receiver at the specified indexes.

Comparing Arrays

- `firstObjectCommonWithArray:` (page 27)
  Returns the first object contained in the receiver that's equal to an object in another given array.
- `isEqualToArray:` (page 41)
  Compares the receiving array to another array.

Deriving New Arrays

- `arrayByAddingObject:` (page 20)
  Returns a new array that is a copy of the receiver with a given object added to the end.
- `arrayByAddingObjectsFromArray:` (page 20)
  Returns a new array that is a copy of the receiver with the objects contained in another array added to the end.
- `filteredArrayUsingPredicate:` (page 26)
  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 51)
  Returns a new array containing the receiver's elements that fall within the limits specified by a given range.

Sorting

- `sortedArrayHint` (page 47)
  Analyzes the receiver and returns a "hint" that speeds the sorting of the array when the hint is supplied to `sortedArrayUsingFunction:context:hint:` (page 49).
- `sortedArrayUsingFunction:context:` (page 48)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingFunction:context:hint:` (page 49)
  Returns a new array that lists the receiver's elements in ascending order as defined by the comparison function *`comparator`*.
- `sortedArrayUsingDescriptors:` (page 47)
  Returns a copy of the receiver sorted as specified by a given array of sort descriptors.
- `sortedArrayUsingSelector:` (page 49)
  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 47)
  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 50)
  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 21)
  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 22)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:` (page 23)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `descriptionWithLocale:indent:` (page 23)
  Returns a string that represents the contents of the receiver, formatted as a property list.
- `writeToFile:atomically:` (page 52)
  Writes the contents of the receiver to a file at a given path.

- `writeToURL:atomically:` (page 52)
  Writes the contents of the receiver to the location specified by a given URL.

Collecting Paths

- `pathsMatchingExtensions:` (page 44)
  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 18)
  Raises an exception.
- `removeObserver:forKeyPath:` (page 45)
  Raises an exception.
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 19)
  Registers *`anObserver`* to receive key value observer notifications for the specified *`keypath`* relative to the objects at *`indexes`*.
- `removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 45)
  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 46)
  Invokes `setValue:forKey:` on each of the receiver's items using the specified *`value`* and *`key`*.
- `valueForKey:` (page 51)
  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 16)
`+ arrayWithObjects:` (page 17)

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 17)
`- initWithObjects:` (page 39)

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 52) 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 52)

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 52) 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 52)

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 14)
`+ arrayWithObjects:` (page 17)

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 14)
`+ arrayWithObject:` (page 16)

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 55)
- `getObjects:range:` (page 27)

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:`.

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

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

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

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 45)
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 19)

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:`.

Availability
Available in iOS 2.0 and later.

See Also
`- removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 45)

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:` (`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:` (`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: (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: 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 31)
- `indexOfObjectIdenticalTo:` (page 35)

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 42)

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 23)
- `descriptionWithLocale:indent:` (page 23)

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 23).

Availability
Available in iOS 2.0 and later.

See Also
- `description` (page 22)
- `descriptionWithLocale:indent:` (page 23)

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 23), 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 22) method.

Availability
Available in iOS 2.0 and later.

See Also
- `- description` (page 22)
- `- descriptionWithLocale:` (page 23)

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 25)
- `makeObjectsPerformSelector:` (page 41)
- `makeObjectsPerformSelector:withObject:` (page 42)

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 26)
- `makeObjectsPerformSelector:` (page 41)
- `makeObjectsPerformSelector:withObject:` (page 42)

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 25)
- `makeObjectsPerformSelector:` (page 41)
- `makeObjectsPerformSelector:withObject:` (page 42)

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:` to check for object equality.

Availability

Available in iOS 2.0 and later.

See Also

- containsObject: (page 21)

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`.

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 18)

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` 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`

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:` returns `YES`.

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

Availability
Available in iOS 2.0 and later.

See Also
- containsObject: (page 21)
- indexOfObjectIdenticalTo: (page 35)

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: returns YES.

Availability
Available in iOS 2.0 and later.

See Also
- containsObject: (page 21)
- indexOfObjectIdenticalTo:inRange: (page 35)

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`.

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`.

opts

Options for the search. For possible values, see "NSBinarySearchingOptions" (page 53).

If you specify both `NSBinarySearchingFirstEqual` (page 53) and `NSBinarySearchingLastEqual` (page 53), 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`.

Return Value

If the `NSBinarySearchingInsertionIndex` (page 54) option is not specified:

- If the *obj* is found and neither `NSBinarySearchingFirstEqual` (page 53) nor `NSBinarySearchingLastEqual` (page 53) is specified, returns an arbitrary matching object's index.
- If the `NSBinarySearchingFirstEqual` (page 53) option is also specified, returns the lowest index of equal objects.
- If the `NSBinarySearchingLastEqual` (page 53) option is also specified, returns the highest index of equal objects.
- If the object is not found, returns `NSNotFound`.

If the `NSBinarySearchingInsertionIndex` (page 54) 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 53) nor `NSBinarySearchingLastEqual` (page 53) is specified, returns any equal or one larger index than any matching object's index.
- If the `NSBinarySearchingFirstEqual` (page 53) option is also specified, returns the lowest index of equal objects.
- If the `NSBinarySearchingLastEqual` (page 53) 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 and/or NSEnumerationReverse 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 21)
- `indexOfObject:` (page 31)

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 21)
- `indexOfObject:inRange:` (page 32)

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 16)
- initWithObjects: (page 39)

Declared In
NSArray.h

initWithArray:copyItems:

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

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

Parameters

array

An array.

flag

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

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
- initWithArray: (page 37)
+ arrayWithObject: (page 16)
- initWithObjects: (page 39)

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 52) 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 15)
`- writeToFile:atomically:` (page 52)

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 52) 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 16)
`- writeToURL:atomically:` (page 52)

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 40)
+ `arrayWithObjects:` (page 17)
- `initWithArray:` (page 37)

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 39)
+ `arrayWithObjects:` (page 17)

- initWithArray: (page 37)

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: 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 (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 42)

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 41)

Declared In

`NSArray.h`

objectAtIndex:

Returns the object located at *index*.

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

Parameters

index

An index within the bounds of the receiver.

Return Value
The object located at *index*.

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

Availability
Available in iOS 2.0 and later.

See Also
- `count` (page 22)
- `objectsAtIndexes:` (page 44)

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 46)

`nextObject` (`NSEnumerator`)

Declared In
`NSArray.h`

objectsAtIndexes:

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

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

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

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

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

Availability
Available in iOS 2.0 and later.

See Also
- `count` (page 22)
- `objectAtIndex:` (page 42)

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 18)
- `removeObserver:fromObjectsAtIndexes:forKeyPath:` (page 45)

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:`.

Availability
Available in iOS 2.0 and later.

See Also
- `addObserver:toObjectsAtIndexes:forKeyPath:options:context:` (page 19)

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 43)
nextObject (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 51)

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 49).

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

Availability
Available in iOS 2.0 and later.

See Also
- `sortedArrayUsingFunction:context:hint:` (page 49)

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 49)
- `sortedArrayUsingFunction:context:` (page 48)
- `sortedArrayUsingFunction:context:hint:` (page 49)

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 47)
- `sortedArrayUsingFunction:context:hint:` (page 49)
- `sortedArrayUsingSelector:` (page 49)

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 48), 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 47). 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 47) 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 49) 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 47)
- sortedArrayUsingFunction:context: (page 48)
- sortedArrayUsingSelector: (page 49)

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:` 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 47)
- `sortedArrayUsingFunction:context:` (page 48)
- `sortedArrayUsingFunction:context:hint:` (page 49)

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 46)

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` 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 15) or the instance method `initWithContentsOfFile:` (page 38). 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 38)

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 16) or the instance method initWithContentsOfURL: (page 39).

Availability

Available in iOS 2.0 and later.

See Also

- initWithContentsOfURL: (page 39)

Declared In

NSArray.h

Constants

NSBinarySearchingOptions

Options for searches and insertions using indexOfObject:inSortedRange:options:usingComparator: (page 32).

```
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`.

Deprecated NSArray Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in iOS 4.0

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 18)

Declared In

`NSArray.h`

Document Revision History

This table describes the changes to *NSArray Class Reference*.

Date	Notes
2010-04-29	Clarified description of indexOfObjectWithOptions:passingTest: method.
2009-11-17	Corrected declarations of several block object parameters.
2009-10-19	Added information to methods that use Blocks with the BOOL *stop parameter.
2008-10-03	Updated for Mac OS X v10.6.
2007-10-31	Added a link to "NSMutableArray Class Reference."
2007-05-03	Added definitions for key-value observing methods.
2007-04-02	Updated for Mac OS X v10.5.
2006-07-24	Clarified description of filteredArrayUsingPredicate:.
2006-05-23	First publication of this content as a separate document.

Document Revision History