NSMutableArray Class Reference

Cocoa > Data Management

2008-11-17

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

Simultaneously published in the United States and Canada.

Contents

NSMutableArray Class Reference 5

Overview 5
Tasks 6
- Creating and Initializing a Mutable Array 6
- Adding Objects 6
- Removing Objects 7
- Replacing Objects 7
- Filtering Content 8
- Rearranging Content 8

Class Methods 8
- arrayWithCapacity: 8

Instance Methods 9
- addObject: 9
- addObjectsFromArray: 9
- exchangeObjectAtIndex:withObjectAtIndex: 10
- filterUsingPredicate: 10
- initWithCapacity: 11
- insertObject:atIndex: 11
- insertObjects:atIndexes: 12
- removeAllObjects 14
- removeLastObject 14
- removeObject: 14
- removeObject:inRange: 15
- removeObjectAtIndex: 16
- removeObjectIdenticalTo: 17
- removeObjectIdenticalTo:inRange: 17
- removeObjectsAtIndexes: 18
- removeObjectsFromIndices:numIndices: 19
- removeObjectsInArray: 19
- removeObjectsInRange: 20
- replaceObjectAtIndex:withObject: 20
- replaceObjectsAtIndexes:withObjects: 21
- replaceObjectsInRange:withObjectsFromArray: 22
- replaceObjectsInRange:withObjectsFromArray:range: 22
- setArray: 23
- sortUsingDescriptors: 23
- sortUsingFunction:context: 24
- sortUsingSelector: 24

Document Revision History 27

Index 29

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 Mac OS X v10.0 and later.
Declared in	NSArray.h NSPredicate.h NSSortDescriptor.h
Companion guides	Collections Programming Topics for Cocoa Key-Value Coding Programming Guide
Related sample code	iSpend Quartz Composer WWDC 2005 TextEdit Sketch-112 StickiesExample TextEditPlus

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 11)
- `removeObjectAtIndex:` (page 16)
- `addObject:` (page 9)
- `removeLastObject` (page 14)
- `replaceObjectAtIndex:withObject:` (page 20)

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` message. When an object is removed from a mutable array, it receives a `release` 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` 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];
```

Mac OS X Note: The `filterUsingPredicate:` (page 10) 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 iPhone OS.

Tasks

Creating and Initializing a Mutable Array

`+ arrayWithCapacity:` (page 8)
: Creates and returns an `NSMutableArray` object with enough allocated memory to initially hold a given number of objects.

`- initWithCapacity:` (page 11)
: Returns an array, initialized with enough memory to initially hold a given number of objects.

Adding Objects

`- addObject:` (page 9)
: Inserts a given object at the end of the receiver.

`- addObjectsFromArray:` (page 9)
: Adds the objects contained in another given array to the end of the receiver's content.

`- insertObject:atIndex:` (page 11)
: Inserts a given object into the receiver's contents at a given index.

`- insertObjects:atIndexes:` (page 12)
: Inserts the objects in in a given array into the receiver at the specified indexes.

Removing Objects

- `removeAllObjects` (page 14)
  Empties the receiver of all its elements.
- `removeLastObject` (page 14)
  Removes the object with the highest-valued index in the receiver
- `removeObject:` (page 14)
  Removes all occurrences in the receiver of a given object.
- `removeObject:inRange:` (page 15)
  Removes all occurrences within a specified range in the receiver of a given object.
- `removeObjectAtIndex:` (page 16)
  Removes the object at *index*.
- `removeObjectsAtIndexes:` (page 18)
  Removes the objects at the specified indexes from the receiver.
- `removeObjectIdenticalTo:` (page 17)
  Removes all occurrences of a given object in the receiver.
- `removeObjectIdenticalTo:inRange:` (page 17)
  Removes all occurrences of *anObject* within the specified range in the receiver.
- `removeObjectsFromIndices:numIndices:` (page 19)
  Removes the specified number of objects from the receiver, beginning at the specified index.
- `removeObjectsInArray:` (page 19)
  Removes from the receiver the objects in another given array.
- `removeObjectsInRange:` (page 20)
  Removes from the receiver each of the objects within a given range.

Replacing Objects

- `replaceObjectAtIndex:withObject:` (page 20)
  Replaces the object at *index* with *anObject*.
- `replaceObjectsAtIndexes:withObjects:` (page 21)
  Replaces the objects in the receiver at specified locations specified with the objects from a given array.
- `replaceObjectsInRange:withObjectsFromArray:range:` (page 22)
  Replaces the objects in the receiver specified by one given range with the objects in another array specified by another range.
- `replaceObjectsInRange:withObjectsFromArray:` (page 22)
  Replaces the objects in the receiver specified by a given range with all of the objects from a given array.
- `setArray:` (page 23)
  Sets the receiver's elements to those in another given array.

Filtering Content

- `filterUsingPredicate:` (page 10)
  Evaluates a given predicate against the receiver's content and leaves only objects that match

Rearranging Content

- `exchangeObjectAtIndex:withObjectAtIndex:` (page 10)
  Exchanges the objects in the receiver at given indices.
- `sortUsingDescriptors:` (page 23)
  Sorts the receiver using a given array of sort descriptors.
- `sortUsingFunction:context:` (page 24)
  Sorts the receiver's elements in ascending order as defined by the comparison function *compare*.
- `sortUsingSelector:` (page 24)
  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 Mac OS X v10.0 and later.

See Also
- `initWithCapacity:` (page 11)

Related Sample Code
EnhancedAudioBurn
Fiendishthngs
QTMetadataEditor
Sketch-112
TimelineToTC

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 Mac OS X v10.0 and later.

See Also
- `addObjectsFromArray:` (page 9)
- `removeObject:` (page 14)
- `setArray:` (page 23)

Related Sample Code
CoreRecipes
Quartz Composer WWDC 2005 TextEdit
Sketch-112
StickiesExample
TextEditPlus

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 Mac OS X v10.0 and later.

See Also
- `setArray:` (page 23)
- `removeObject:` (page 14)

Related Sample Code
Quartz Composer WWDC 2005 TextEdit
SimpleCalendar
Sketch-112
StickiesExample
TextEditPlus

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 Mac OS X v10.2 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 Mac OS X v10.4 and later.

See Also
- `filteredArrayUsingPredicate:` (`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 Mac OS X v10.0 and later.

See Also

+ arrayWithCapacity: (page 8)

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 Mac OS X v10.0 and later.

See Also
- `removeObjectAtIndex:` (page 16)

Related Sample Code
SimpleCocoaMovie
SimpleCocoaMovieQT
SimpleScriptingObjects
ThreadsExporter
WhackedTV

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)`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- insertObject:atIndex:` (page 11)

Declared In
`NSArray.h`

removeAllObjects

Empties the receiver of all its elements.

```
- (void)removeAllObjects
```

Availability
Available in Mac OS X v10.0 and later.

See Also
- `removeObject:` (page 14)
- `removeLastObject` (page 14)
- `removeObjectAtIndex:` (page 16)
- `removeObjectIdenticalTo:` (page 17)

Related Sample Code
ABPresence
WhackedTV

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 Mac OS X v10.0 and later.

See Also
- `removeAllObjects` (page 14)
- `removeObject:` (page 14)
- `removeObjectAtIndex:` (page 16)
- `removeObjectIdenticalTo:` (page 17)

Related Sample Code
WhackedTV

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:` to locate matches and then removes them by using `removeObjectAtIndex:` (page 16). 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 Mac OS X v10.0 and later.

See Also

- `removeAllObjects` (page 14)
- `removeLastObject` (page 14)
- `removeObjectAtIndex:` (page 16)
- `removeObjectIdenticalTo:` (page 17)
- `removeObjectsInArray:` (page 19)

Related Sample Code

CoreRecipes

GLChildWindowDemo

Squiggles

WhackedTV

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 Mac OS X v10.0 and later.

See Also
- `removeAllObjects` (page 14)
- `removeLastObject` (page 14)
- `removeObjectAtIndex:` (page 16)
- `removeObjectIdenticalTo:` (page 17)
- `removeObjectsInArray:` (page 19)

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 Mac OS X v10.0 and later.

See Also
- `insertObject:atIndex:` (page 11)
- `removeAllObjects` (page 14)
- `removeLastObject` (page 14)
- `removeObject:` (page 14)
- `removeObjectIdenticalTo:` (page 17)
- `removeObjectsFromIndices:numIndices:` (page 19)

Related Sample Code
EnhancedAudioBurn
EnhancedDataBurn
ImageBackground
QTKitMovieShuffler
SimpleScriptingObjects

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:` method to locate matches and then removes them by using `removeObjectAtIndex:` (page 16). 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 Mac OS X v10.0 and later.

See Also

- `removeAllObjects` (page 14)
- `removeLastObject` (page 14)
- `removeObject:` (page 14)
- `removeObjectAtIndex:` (page 16)

Related Sample Code

EnhancedDataBurn

ImageBackground

QTKitMovieShuffler

TrackBall

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:` method to locate matches and then removes them by using `removeObjectAtIndex:` (page 16). 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 Mac OS X v10.0 and later.

See Also
- `removeAllObjects` (page 14)
- `removeLastObject` (page 14)
- `removeObject:` (page 14)
- `removeObjectAtIndex:` (page 16)
- `removeObjectsAtIndexes:` (page 18)

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 16), but allows you to efficiently remove multiple objects with a single operation. *indexes* specifies the locations of objects to be removed given the state of the receiver when the method is invoked, as illustrated in the following example.

```
NSMutableArray *array = [NSMutableArray arrayWithObjects: @"one", @"a", @"two",
 @"b", @"three", @"four", nil];
NSMutableIndexSet *indexes = [NSMutableIndexSet indexSetWithIndex:1];
[indexes addIndex:3];
[array removeObjectsAtIndexes:indexes];
NSLog(@"array: %@", array);

// Output: array: (one, two, three, four)
```

Availability
Available in Mac OS X v10.4 and later.

See Also
- `initWithCapacity:` (page 11)
- `removeObjectAtIndex:` (page 16)
- `removeObject:inRange:` (page 15)

Declared In
`NSArray.h`

removeObjectsFromIndices:numIndices:

Removes the specified number of objects from the receiver, beginning at the specified index.

```
- (void)removeObjectsFromIndices:(NSUInteger *)indices numIndices:(NSUInteger)count
```

Parameters

indices
A C array of the indices of the objects to remove from the receiver.

count
The number of objects to remove from the receiver.

Discussion
This method is similar to `removeObjectAtIndex:` (page 16), 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 Mac OS X v10.0 and later.

See Also
- `initWithCapacity:` (page 11)
- `removeObjectAtIndex:` (page 16)
- `removeObject:inRange:` (page 15)
- `removeObjectsAtIndexes:` (page 18)

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 14), 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 Mac OS X v10.0 and later.

See Also
- `removeAllObjects` (page 14)
- `removeObjectIdenticalTo:` (page 17)
- `removeObjectsAtIndexes:` (page 18)

Related Sample Code
QTKitAdvancedDocument
SimpleCalendar
StickiesExample

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 16).

Availability
Available in Mac OS X v10.0 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 Mac OS X v10.0 and later.

See Also

- `insertObject:atIndex:` (page 11)
- `removeObjectAtIndex:` (page 16)
- `removeObjectsAtIndexes:` (page 18)
- `replaceObjectsAtIndexes:withObjects:` (page 21)

Related Sample Code

ABPresence

TrackBall

Declared In

`NSArray.h`

replaceObjectsAtIndexes:withObjects:

Replaces the objects in the receiver at specified locations specified with the objects from a given array.

```
- (void)replaceObjectsAtIndexes:(NSIndexSet *)indexes  withObjects:(NSArray *)objects
```

Parameters

indexes

The indexes of the objects to be replaced.

objects

The objects with which to replace the objects in the receiver at the indexes specified by *indexes*. The count of locations in *indexes* must equal the count of *objects*.

Discussion

The indexes in *indexes* are used in the same order as the objects in *objects*.

Availability

Available in Mac OS X v10.4 and later.

See Also

- `insertObject:atIndex:` (page 11)
- `removeObjectAtIndex:` (page 16)

- replaceObjectAtIndex:withObject: (page 20)

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 Mac OS X v10.0 and later.

See Also
- insertObject:atIndex: (page 11)
- removeObjectAtIndex: (page 16)
- replaceObjectAtIndex:withObject: (page 20)
- replaceObjectsAtIndexes:withObjects: (page 21)

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 Mac OS X v10.0 and later.

See Also
- `insertObject:atIndex:` (page 11)
- `removeObjectAtIndex:` (page 16)
- `replaceObjectAtIndex:withObject:` (page 20)
- `replaceObjectsAtIndexes:withObjects:` (page 21)

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 Mac OS X v10.0 and later.

See Also
- `addObjectsFromArray:` (page 9)
- `insertObject:atIndex:` (page 11)

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 Mac OS X v10.3 and later.

See Also
- `sortUsingFunction:context:` (page 24)
- `sortUsingSelector:` (page 24)
- `sortedArrayUsingDescriptors:` (`NSArray`)

Related Sample Code
CoreRecipes

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 Mac OS X v10.0 and later.

See Also
- `sortUsingDescriptors:` (page 23)
- `sortUsingSelector:` (page 24)
- `sortedArrayUsingFunction:context:` (`NSArray`)

Related Sample Code
Reminders

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 Mac OS X v10.0 and later.

See Also

- `sortUsingDescriptors:` (page 23)
- `sortUsingFunction:context:` (page 24)
- `sortedArrayUsingSelector:` (`NSArray`)

Related Sample Code

ABPresence

SearchField

Declared In

`NSArray.h`

Document Revision History

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

Date	Notes
2008-11-17	Noted in overview that predicates are not available on iPhone OS.
2008-10-15	Corrected parameter declaration for sortUsingFunction:.
2007-04-03	Clarified description of primitive methods.
2007-03-07	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a separate document.

Document Revision History

Index

A

addObject: instance method 9
addObjectsFromArray: instance method 9
arrayWithCapacity: class method 8

E

exchangeObjectAtIndex:withObjectAtIndex: instance method 10

F

filterUsingPredicate: instance method 10

I

initWithCapacity: instance method 11
insertObject:atIndex: instance method 11
insertObjects:atIndexes: instance method 12

R

removeAllObjects instance method 14
removeLastObject instance method 14
removeObjectAtIndex: instance method 16
removeObject: instance method 14
removeObject:inRange: instance method 15
removeObjectIdenticalTo: instance method 17
removeObjectIdenticalTo:inRange: instance method 17
removeObjectsAtIndexes: instance method 18
removeObjectsFromIndices:numIndices: instance method 19
removeObjectsInArray: instance method 19
removeObjectsInRange: instance method 20
replaceObjectAtIndex:withObject: instance method 20
replaceObjectsAtIndexes:withObjects: instance method 21
replaceObjectsInRange:withObjectsFromArray: instance method 22
replaceObjectsInRange:withObjectsFromArray:range: instance method 22

S

setArray: instance method 23
sortUsingDescriptors: instance method 23
sortUsingFunction:context: instance method 24
sortUsingSelector: instance method 24