NSArray Class Cluster

Class Cluster Description

NSArray objects manage arrays of objects. The cluster's two public classes, NSArray and NSMutableArray, declare the programmatic interface for static and dynamic arrays, respectively.

The objects you create using these classes are referred to as *array objects*. Because of the nature of class clusters, array objects are not actual instances of the NSArray or NSMutableArray classes but of one of their private subclasses. Although an array object's class is private, its interface is public, as declared by these abstract superclasses, NSArray and NSMutableArray. (See "Class Clusters" in the introduction to the Foundation Kit for more information on class clusters and creating subclasses within a cluster.)

Generally, you instantiate an array object by sending one of the **array...** messages to either the NSArray or NSMutableArray class object. These methods return an array object containing the elements you pass in as arguments. (Note that arrays can't contain the **nil** object.) In general, objects that you add to an array aren't copied; rather, each object receives a **retain** message before its **id** is added to the array. When an object is removed from an array, it's sent a **release** message.

The NSArray class adopts the NSCopying and NSMutableCopying protocols, making it convenient to convert an array of one type to the other.

▶ NSArray

Inherits From:	NSObject
Conforms To:	NSCopying NSMutableCopying NSObject (NSObject)
Declared In:	foundation/NSArray.h

Class Description

The NSArray class declares the programmatic interface to an object that manages an immutable array of objects. NSArray's two primitive methods—**count** and **objectAtIndex:**—provide the basis for all the 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** and **reverseObjectEnumerator** also permit sequential access of the elements of the array, differing only in the direction of travel through the elements. These methods are provided so that array objects can be traversed in a manner similar to that used for objects of other collection classes, such as NSDictionary.

NSArray provides methods for querying the elements of the array. **indexOfObject:** 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:**, is provided for the less common case of determining whether a specific object is present in the array. **indexOfObjectIdenticalTo:** tests each element in the array to see whether its **id** matches that of the argument.

NSArray's **makeObjectsPerform:** and **makeObjectsPerform:withObject:** methods let you act on the individual objects in the array by sending them messages. To act on the array as a whole, a variety of methods are defined. You can create a sorted version of the array (**sortedArrayUsingSelector:** and **sortedArrayUsingFunction:context:**), extract a subset of the array (**subarrayWithRange:**), or concatenate the elements of an array of NSString objects into a single string (**componentsJoinedByString:**). In addition, you can compare two array objects using the **isEqualToArray:** and **firstObjectCommonWithArray:** methods.

Instance Variables

None declared in this class.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copy – copyWithZone:
NSMutableCopying	– mutableCopy – mutableCopyWithZone:

Method Types

Allocating and initializing an NSArray	+ allocWithZone: + array + arrayWithObject: + arrayWithObjects: – initWithArray: – initWithObjects: – initWithObjects:count:
Querying the array	– containsObject: – count – indexOfObject: – indexOfObjectIdenticalTo: – lastObject – objectAtIndex: – objectEnumerator – reverseObjectEnumerator
Sending messages to elements	– makeObjectsPerform: – makeObjectsPerform:withObject:
Comparing arrays	– firstObjectCommonWithArray: – isEqualToArray:
Deriving new arrays	– sortedArrayUsingFunction:context: – sortedArrayUsingSelector: – subarrayWithRange:

Joining string elements – componentsJoinedByString:

Creating a description of the array
– description
– descriptionWithIndent:

Class Methods

allocWithZone:

+ **allocWithZone:**(NSZone *)*zone*

Creates and returns an uninitialized array object in the specified zone. If the receiver is the NSArray class object, an instance of an appropriate immutable private subclass is returned; otherwise, an object of the receiver's class is returned.

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

See also: + **array**, + **arrayWithObject:**, + **arrayWithObjects:**

array

+ **array**

Creates and returns an empty array object. This method is declared primarily for the use of mutable subclasses of NSArray.

See also: + **arrayWithObject:**, + **arrayWithObjects:**

arrayWithObject:

+ **arrayWithObject:***anObject*

Creates and returns an array object containing the single element *anObject*.

See also: + **array**, + **arrayWithObjects:**

arrayWithObjects:

+ arrayWithObjects:*firstObj*, ...

Creates and returns an array object containing the objects in the argument list. The argument list is a comma-separated list of objects ending with **nil**.

As an example, this excerpt creates an array object containing three different types of elements (assuming *aPath* exits):

```
NSArray *myArray;
NSData *someData = [NSData dataWithContentsOfFile:aPath];
NSValue *aValue = [NSNumber numberWithInt:5];
NSString *aString = @”a string”;

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

See also: + **array**, + **arrayWithObject:**

Instance Methods

componentsJoinedByString:

– (NSString *)**componentsJoinedByString:**(NSString *)*separator*

Constructs and returns an NSString object that is the result of interposing *separator* between the elements of the receiver’s array. For example, this code excerpt causes *myTextObject* to display the path **/NextDeveloper/Examples** (assuming *stream* exists):

```
NSArray *pathArray = [NSArray arrayWithObjects:@"NextDeveloper",
    @"Examples", nil];
NSLog("The path is /%@.\n”,
    [pathArray componentsJoinedByString:@”/”]);
[myTextObject readText:stream];
```

Each element of the receiver’s array must be an NSString or an error occurs. If the receiver has no elements, an NSString object representing an empty string is returned.

See also: – **componentsSeparatedByString:** (NSString)

containsObject:

– (BOOL)**containsObject:***anObject*

Returns YES if *anObject* is present in the array. This method works by invoking **indexOfObject:**, which compares each object in the array to *anObject* by sending them each an **isEqual:** message.

See also: – **indexOfObject:**, – **indexOfObjectIdenticalTo:**, – **isEqual:** (NSObject protocol)

count

– (unsigned)**count**

Returns the number of objects currently in the array.

See also: – **objectAtIndex:**

description

– (NSString *)**description**

Returns a string object that represents the contents of the receiver. The returned object uses the PropertyList format.

See also: **descriptionWithIndent:**

descriptionWithIndent:

– (NSString *)**descriptionWithIndent:**(unsigned)*level*

Returns a string object that represents the contents of the receiver. The returned object uses the PropertyList format. *level* allows you to specify a level of indent, to make the output more readable: set *level* to **0** for no indent, or **1** to have the output indented four spaces.

See also: **description**

firstObjectCommonWithArray:

– **firstObjectCommonWithArray:**(NSArray *)*otherArray*

Returns the first object from the receiver's array that's equal to an object in *otherArray.* This method invokes **containsObject:** to determine whether objects are equal. If no such object is found, this method returns **nil**.

See also: – **containsObject:**, – **isEqual:** (NSObject protocol)

hash

@protocol NSObject
– (unsigned int)**hash**

Returns an unsigned integer that can be used as a table address in a hash table structure. For an array object, **hash** returns the number of elements in the array. If two array objects are equal (as determined by the **isEqual:** method), they will have the same hash value.

See also: – **isEqual:**

indexOfObject:

– (unsigned)**indexOfObject:***anObject*

Returns the index of *anObject*, if found; otherwise, returns NSNotFound. This method checks the elements in the array from last to first by sending them an **isEqual:** message.

See also: – **containsObject:**, – **indexOfObjectIdenticalTo:**, – **isEqual:** (NSObject protocol)

indexOfObjectIdenticalTo:

– (unsigned)**indexOfObjectIdenticalTo:***anObject*

Returns the index of *anObject*, if found; otherwise, returns NSNotFound. This method checks the elements in the array from last to first by comparing their **id**s.

See also: – **containsObject:**, – **indexOfObject:**, – **isEqual:** (NSObject protocol)

initWithArray:

– **initWithArray:**(NSArray *)*array*

Initializes a newly allocated array object by placing in it the objects contained in *array.* After an immutable array has been initialized in this way, it can't be modified. Returns **self**.

See also: **+ arrayWithObject:**, – **initWithArray:copyItems:**, – **initWithObjects:**

initWithObjects:

– **initWithObjects:***firstObj*, ...

Initializes a newly allocated array object by placing in it the objects in the argument list. This list is a comma-separated list of objects ending with **nil**. This method invokes **initWithObjects:count:** as part of its implementation; thus, the objects are retained rather than copied as they're added to the array. After an immutable array has been initialized in this way, it can't be modified. Returns **self**.

See also: – **initWithObjects:count:**, **+ arrayWithObjects:**, – **initWithArray:**, – **initWithArray:copyItems:**

initWithObjects:count:

– **initWithObjects:**(id *)*objects* **count:**(unsigned)*count*

Initializes a newly allocated array object by placing in it *count* objects from the *objects* array. Each object in the *objects* array receives a **retain** message as it's added to the array. After an immutable array has been initialized in this way, it can't be modified. Returns **self**.

See also: – **initWithObjects:**, **+ arrayWithObjects:**, – **initWithArray:**, – **initWithArray:copyItems:**

isEqual:

@protocol NSObject
– (BOOL)**isEqual:***anObject*

Returns YES if the receiver and *anObject* are equal; otherwise returns NO. A YES return value indicates that the receiver and *anObject* both inherit from NSArray and contain the same data (as determined by the **isEqualToArray:** method).

See also: – **isEqualToArray:**

isEqualToArray:

– (BOOL)**isEqualToArray:**(NSArray *)*otherArray*

Compares the receiving array object to *otherArray.* If the contents of *otherArray* are equal to the contents of the receiver, this method returns YES. If not, it returns NO.

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.

See also: – **isEqual:** (NSObject protocol)

lastObject

– **lastObject**

Returns the last object in the array by invoking the **objectAtIndex:** and **count** methods. If the array is empty, **lastObject** raises an NSRangeException error.

See also: – **removeLastObject**

makeObjectsPerform:

– (void)**makeObjectsPerform:**(SEL)*aSelector*

Sends an *aSelector* message to each object in the array in reverse order (starting with the last object and continuing backwards through the array to the first object). The *aSelector* method must be one that takes no arguments. It shouldn't have the side effect of modifying the receiver. The messages are sent using the **perform:** method declared in the NSObject protocol.

See also: – **makeObjectsPerform:withObject:**, – **perform:** (NSObject protocol)

makeObjectsPerform:withObject:

– (void)**makeObjectsPerform:**(SEL)*aSelector* **withObject:***anObject*

Sends an *aSelector* message to each object in the array in reverse order (starting with the last object and continuing backwards through the array to the first object). The message is sent each time with *anObject* as an argument, so the *aSelector* method must be one that takes a single argument of type **id**. The *aSelector* method shouldn't, as a side effect, modify the receiver. The messages are sent using the **perform:with:** method declared in the NSObject protocol.

See also: – **makeObjectsPerform:**, – **perform:with:** (NSObject protocol)

objectAtIndex:

– **objectAtIndex:**(unsigned)*index*

Returns the object located at *index*. If *index* is beyond the end of the array, an NSRangeException error is raised.

This method is fast, with an execution time that's independent of the number of objects in the array.

See also: – **count**

objectEnumerator

– (NSEnumerator *)**objectEnumerator**

Returns an enumerator object that lets you access each object in the array, starting with the first element.

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

When this method is used with mutable subclasses of NSArray, your code shouldn't modify the array during enumeration.

See also: – **reverseObjectEnumerator,** – **nextObject** (NSEnumerator protocol)

reverseObjectEnumerator

– (NSEnumerator *)**reverseObjectEnumerator**

Returns an enumerator object that lets you access each object in the array, from the last element to the first. Your code shouldn't modify the array during enumeration.

See also: – **objectEnumerator,** – **nextObject** (NSEnumerator protocol)

sortedArrayUsingFunction:context:

– (NSArray *)**sortedArrayUsingFunction:**(int(*)(id, id, void *))*comparator*
context:(void *)*context*

Returns an array object that lists the receiver's elements in ascending order as defined by the comparison function *comparator.* The new array contains references to the receiver's

elements, not copies of them. The retain count is incremented for each element in the receiving array.

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 integer number objects) and a comparison function of this type,

```
int intSort(id num1, id num2, void *context)
{
    int v1 = [num1 intValue];
    int v2 = [num2 intValue];
    return v1-v2;
}
```

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

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

Sorting using this method guarantees no more than N log N comparisons.

See also: – **sortedArrayUsingSelector:**

sortedArrayUsingSelector:

– (NSArray *)**sortedArrayUsingSelector:**(SEL)*comparator*

Returns an array object that lists the receiver's elements in ascending order, as determined by the comparison method specified by the selector *comparator.* The new array contains references to the receiver's elements, not copies of them. The retain count is incremented for each element in the receiving array.

The *comparator* message is sent to each object in the array, and has as its single argument another object in the array. The comparator method is used to compare two elements at a time and 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. Sorting using this method guarantees no more than N log N comparisons.

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

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

See also: – **sortedArrayUsingFunction:context:**

subarrayWithRange:

– (NSArray *)**subarrayWithRange:**(NSRange)*range*

Returns an array object containing the receiver's elements that fall within the limits specified by *range*. If *range* isn't within the receiver's range of elements, an NSRangeException error is raised. The retain count is incremented for each element in the receiving array.

For example, this excerpt creates an array containing the elements found in the first half of *wholeArray*, which is assumed to exist.

```
NSArray *halfArray;
NSRange theRange;

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

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

▶ NSMutableArray

Inherits From: NSArray

Conforms To: NSCoding
NSCopying
NSMutableCopying
NSObject (NSObject)

Declared In: foundation/NSArray.h

Class Description

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 it inherits from NSArray.

The array operations that NSMutableArray declares are conceptually based on these three methods:

addObject:
replaceObjectAtIndex:withObject:
removeLastObject

The other methods in its interface provide convenient ways of inserting an object into a specific slot in the array and of removing an object based on its identity or position in the array.

When an object is removed from a mutable array it receives a **release** message. If there are no further references to the object, it's deallocated. Note that if your program keeps a reference to such an object, the reference will become invalid unless you remember to send the object a **retain** message before it's removed from the array. For example, the third statement below could result in a run-time error, except for the **retain** message in the first statement:

```
id anObject = [[anArray objectAtIndex:0] retain];
[anArray removeObjectAtIndex:0];
[anObject someMessage];
```

A Note for Those Creating Subclasses of NSMutableArray

Although conceptually the NSMutableArray class has three primitive methods, two others also access the array object's data directly. These methods are:

insertObject:atIndex:
removeObjectAtIndex:

These methods could be implemented using the primitives listed above but in doing so would incur unnecessary overhead from the **retain** and **release** messages that objects would receive as they are shifted to accommodate the insertion or deletion of an element.

Instance Variables

None declared in this class.

Method Types

Allocating and initializing an NSMutableArray	+ allocWithZone: + arrayWithCapacity: – initWithCapacity:
Adding objects	– addObject: – addObjectsFromArray: – insertObject:atIndex:
Removing objects	– removeAllObjects – removeLastObject – removeObject: – removeObjectAtIndex: – removeObjectIdenticalTo: – removeObjectsFromIndices:numIndices: – removeObjectsInArray:
Replacing objects	– replaceObjectAtIndex:withObject:
Rearranging objects	– sortUsingFunction:context:

Class Methods

allocWithZone:

+ **allocWithZone:**(NSZone *)*zone*

Creates and returns an uninitialized NSMutableArray object in the specified zone. If the receiver is the NSMutableArray class object, an instance of an appropriate private subclass is returned; otherwise, an object of the receiver's class is returned.

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

See also: + **arrayWithCapacity:**

arrayWithCapacity:

+ **arrayWithCapacity:**(unsigned)*numItems*

Creates and returns an NSMutableArray object, giving it enough allocated memory to hold *numItems* objects. NSMutableArray objects allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

See also: – **initWithCapacity:**

Instance Methods

addObject:

– (void)**addObject:***anObject*

Inserts *anObject* at the end of the NSMutableArray. The object receives a **retain** message as it's added to the array. If *anObject* is **nil**, an NSInvalidArgumentException error occurs.

This method is fast, with an execution time that's independent of the number of objects in the array.

See also: – **addObjectsFromArray:**, – **removeObject:**

addObjectsFromArray:

– (void)**addObjectsFromArray:**(NSArray *)*otherArray*

Adds the objects contained in *otherArray* to the end of the receiver's array of objects. This method invokes **addObject:** as part of its implementation. Its execution speed is proportional to the number of elements in *otherArray.*

See also: – **addObject:**

copyWithZone:

@protocol NSCopying
– **copyWithZone:**(NSZone *)*aZone*

Creates and returns an immutable copy of the receiver. The new array object contains copies of the receiver's elements.

Note that it's your responsibility to release an array object created in this way.

See also: – **mutableCopyWithZone:** (NSObject protocol)

initWithCapacity:

– **initWithCapacity:**(unsigned)*numItems*

Initializes a newly allocated array object, giving it enough memory to hold *numItems* objects. Mutable array objects allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity. Returns **self**.

See also: – **init**, – **arrayWithCapacity:**

insertObject:atIndex:

– (void)**insertObject:***anObject* **atIndex:**(unsigned)*index*

Inserts *anObject* into the NSMutableArray at *index*. If *index* is already occupied, the objects at *index* and beyond are shifted down one slot to make room. *anObject* receives a **retain** message as it's added to the array. This method raises an NSInvalidArgumentException error if *anObject* is **nil** and an NSRangeException error if *index* is beyond the end of the array.

This method is slow, with an execution time that's dependent on the number of objects in the array.

See also: – **removeObjectAtIndex:**

removeAllObjects

– (void)**removeAllObjects**

Empties the NSMutableArray of all its elements. As each object is removed, it's sent a **release** message.

See also: – **removeObject:**, – **removeLastObject**, – **removeObjectAtIndex:**, – **removeObjectIdenticalTo:**

removeLastObject

– (void)**removeLastObject**

Removes the last object in the array and sends it a **release** message. **removeLastObject:** raises an NSRangeException error if there are no objects in the array.

This method is fast, with an execution time that's independent of the number of objects in the array.

See also: – **removeAllObjects**, – **removeObject:**, – **removeObjectAtIndex:**, – **removeObjectIdenticalTo:**

removeObject:

– (void)**removeObject:***anObject*

Removes all occurrences of *anObject* in the array. This method uses the **indexOfObject:** method to locate matches and then removes them by invoking **removeObjectAtIndex:**. Thus, matches are determined on the basis of an object's response to an **isEqual:** message.

See also: – **removeAllObjects**, – **removeLastObject**, – **removeObjectAtIndex:**, – **removeObjectIdenticalTo:**

removeObjectAtIndex:

– (void)**removeObjectAtIndex:**(unsigned)*index*

Removes the object at *index* and moves all elements beyond *index* up one slot to fill the gap. The removed object receives a **release** message. **removeObjectAtIndex:** raises an NSRangeException error if *index* is beyond the end of the array.

This method is slow, with an execution time that's dependent on the number of objects in the array.

See also: – **removeAllObjects**, – **removeLastObject**, – **removeObject:**, – **removeObjectIdenticalTo:, – removeObjectsFromIndices:numIndices:**

removeObjectIdenticalTo:

– (void)**removeObjectIdenticalTo:***anObject*

Removes all occurrences of *anObject* in the array. This method uses the **indexOfObjectIdenticalTo:** method to locate matches and then removes them by invoking **removeObjectAtIndex:**. Thus, matches are determined on the basis of an object's **id**.

See also: – **removeAllObjects**, – **removeLastObject**, – **removeObject:**, – **removeObjectAtIndex:**

removeObjectsFromIndices:numIndices:

– (void)**removeObjectsFromIndices:**(unsigned *)*indices*
numIndices:(unsigned)*count*

This method is similar to **removeObjectAtIndex:**, but allows you to efficiently remove multiple objects with a single operation. *count* indicates the number of objects to be removed, while *indices* points to the first in a list of indexes.

This method does not distribute and therefore should be used sparingly.

See also: – **removeObjectAtIndex:**, – **removeAllObjects**

removeObjectsInArray:

– (void)**removeObjectsInArray:**(NSArray *)*otherArray*

This method is similar to **removeObject:**, but allows you to efficiently remove large sets of objects with a single operation. It assumes that all elements in *otherArray*—which are the objects to be removed—respond to **hash** and **isEqual:**.

This method does not distribute and therefore should be used sparingly.

See also: – **removeAllObjects**, – **removeObject:**, – **removeObjectIdenticalTo:**

replaceObjectAtIndex:withObject:

– (void)**replaceObjectAtIndex:**(unsigned)*index* **withObject:***anObject*

Replaces the object at *index* with *anObject*. *anObject* receives a **retain** message as it's added to the array, and the previous object at *index* receives a **release** message. This method raises an NSInvalidArgumentException error if *anObject* is **nil** and an NSRangeException error if *index* is beyond the end of the array.

This method is fast, with an execution time that's independent of the number of objects in the array.

See also: – **insertObjectAtIndex:**, – **removeObjectAtIndex:**

sortUsingFunction:context:

– (void)**sortUsingFunction:**(int (*)(id ,id ,void *))*compare* **context:**(void *)*context*

Sorts (in place) the receiver's elements in ascending order as defined by the comparison function *compare*. *context* is passed to the comparator function as its third argument.