NSSet Class Cluster

Class Cluster Description

The NSSet, NSMutableSet, and NSCountedSet classes declare the programmatic interface for objects that store unordered sets of objects.

Because of the nature of class clusters, the objects you create with the NSSet class cluster are not actual instances of NSSet or NSMutableSet. Rather, the instances belong to one of their private subclasses. (For convenience, we use the term *set* to refer to any one of these instances without specifying its exact class membership.) Although a set's class is private, its interface is public, as declared by the abstract superclasses NSSet and NSMutableSet.Note that NSCountedSet is not part of the class cluster; it is a concrete subclass of NSMutableSet.

NSSet declares the programmatic interface for static sets of objects. You establish a static set's entries when it's created, and thereafter the entries can't be modified. NSMutableSet, on the other hand, declares a programmatic interface for dynamic sets of objects. A dynamic—or mutable—set allows the addition and deletion of entries at any time, automatically allocating memory as needed.

Use sets as an alternative to arrays when the order of elements isn't important and performance in testing whether an object is contained in the set *is* a consideration—while arrays are ordered, testing for membership is slower than with sets.

Objects in a set must respond to the NSObject protocol methods **hash** and **isEqual:**. See the NSObject protocol for more information.

Note: If mutable objects are stored in a set, either the **hash** method of the objects shouldn't depend on the internal state of the mutable objects or the mutable objects shouldn't be modified while they're in the set (note that it can be difficult to know whether or not a given object is in a collection).

Objects added to a set are not copied; rather, each object receives a **retain** message before it's added to a set.

Generally, you create a temporary set by sending one of the **set…** methods to the NSSet class object. These methods return an NSSet object containing the elements (if any) you pass in as arguments. The **set** method is a "convenience" method to create an empty mutable set. Newly created instances of NSSet created by invoking the **allocWithZone:** method can be populated with objects using any of the **init…** methods.

The set classes adopt the NSCopying and NSMutableCopying protocols, making it convenient to convert a set of one type to the other.

NSSet

Inherits From:	NSObject
Conforms To:	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Declared In:	Foundation/NSSet.h

Class at a Glance

Purpose

An NSSet object stores an immutable set of objects.

Principal Attributes

- The objects that make up the set.

Creation

+ set	Returns an empty set.
+ setWithArray:	Returns a set containing a number of objects from an array.
+ setWithObject:	Returns a set containing a single object.
+ setWithObjects:	Returns a set containing a number of objects.
+ setWithObjects:count:	Returns a set containing a specified number of objects.
+ setWithSet:	Returns a set containing a number of objects from another set.

Commonly Used Methods

– allObjects	Returns an array containing the set's member objects.
– count	Returns the number of objects in the set.
– containsObject:	Indicates whether a given object is present in the set.

Primitive Methods

– count
– member:
– objectEnumerator

Class Description

The NSSet class declares the programmatic interface to an object that manages an immutable set of objects. NSSet provides support for the mathematical concept of a *set*. A set, both in its mathematical sense and in the implementation of NSSet, is an *unordered* collection of distinct elements. The NSMutableSet and NSCountedSet classes are provided for sets whose contents may be altered.

NSSet provides methods for querying the elements of the set. **allObjects** returns an array containing the objects in a set. **anyObject** returns some object in the set. **count** returns the number of objects currently in the set. **member:** returns the object in the set that is equal to a specified object. Additionally, the **intersectsSet:** tests for set intersection, **isEqualToSet:** tests for set equality, and **isSubsetOfSet:** tests for one set being a subset of another.

The **objectEnumerator** method provides for traversing elements of the set one by one.

NSSet's **makeObjectsPerform:** and **makeObjectsPerform:withObject:** methods provides for sending messages to individual objects in the set.

Exceptions

NSSet implements the **encodeWithCoder:** method, which raises NSInternalInconsistencyException if the number of objects enumerated for encoding turns out to be unequal to the number of objects in the set.

Adopted Protocols

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

Method Types

Creating a set	+ allocWithZone: + set + setWithArray: + setWithObject: + setWithObjects: – initWithArray: – initWithObjects: – initWithObjects:count: – initWithSet: – initWithSet:copyItems:
Counting entries	– count
Accessing the members	– allObjects – anyObject – containsObject: – makeObjectsPerform: – makeObjectsPerform:withObject: – member: – objectEnumerator
Comparing sets	– isSubsetOfSet: – intersectsSet: – isEqualToSet:
Describing a set	– description – descriptionWithLocale:

Class Methods

allocWithZone:

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

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

Typically, you create temporary sets using the **set...** class methods, not the **allocWithZone:** and **init...** methods. Note that it's your responsibility to free objects created with the **allocWithZone:** method.

See also: **+ set**, **+ setWithObject:**, **+ setWithObjects:**, **+ setWithArray:**

set

+ (id)**set**

Creates and returns an empty set. This method is declared primarily for the use of mutable subclasses of NSSet.

See also: **+ setWithArray:**, **+ setWithObject:**, **+ setWithObjects:**

setWithArray:

+ (id)**setWithArray:**(NSArray *)*anArray*

Creates and returns a set containing those objects contained within the array *anArray.*

See also: **+ set**, **+ setWithObject:**, **+ setWithObjects:**

setWithObject:

+ (id)**setWithObject:**(id)*anObject*

Creates and returns a set containing a single member, *anObject*. *anObject* receives a **retain** message after being added to the set.

See also: **+ setWithArray:**, **+ set**, **+ setWithObjects:**

setWithObjects:

+ (id)**setWithObjects:**(id)*anObject, ...*

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

As an example, the following code excerpt creates a set containing three different types of elements (assuming *aPath* exits):

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

mySet = [NSSet setWithObjects:someData, aValue, aString, nil];
```

See also: **+ setWithArray:**, **+ set**, **+ setWithObject:**

setWithObjects:count:

+ (id)**setWithObjects:**(id *)*objects* **count:**(unsigned int)*count*

Creates and returns a set containing *count* objects from the list of objects specified by *objects*.

setWithSet:

+ (id)**setWithArray:**(NSSet *)*aSet*

Creates and returns a set containing those objects contained within the set *aSet*.

nstance Methods

allObjects

– (NSArray *)**allObjects**

Returns an array containing the receiver's members, or an empty array if the receiver has no members. The order of the objects in the array isn't defined.

anyObject

– (id)**anyObject**

Returns one of the objects in the set (essentially chosen at random), or **nil** if the set contains no objects.

See also: – **allObjects**, – **objectEnumerator**

containsObject:

– (BOOL)**containsObject:**(id)*anObject*

Returns YES if *anObject* is present in the set, NO otherwise.

See also: – **member:**

count

– (unsigned int)**count**

Returns the number of members in the set.

description

– (NSString *)**description**

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

See also: – **descriptionWithLocale:**

descriptionWithLocale:

– (NSString *)**descriptionWithLocale:**(NSDictionary *)*locale*

Returns a string object that represents the contents of the receiver, formatted as a property list. *locale* specifies options used for formatting each of the receiver's members, each of which is sent **descriptionWithLocale:**with *locale* passed along as the sole parameter. (If the receiver's members do not respond to **descriptionWithLocale:**, this method sends **description** instead.) If you do not want the receiver's members to be formatted, specify **nil** for *locale*.

See also: – **description**

hash

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

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

See also: – **isEqual:**

initWithArray:

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

Initializes a newly allocated set with the objects that are contained in *array*. This method steps through *array*, adding members to the new set as it goes. Each object receives a **retain** message as it is added to the set. Returns **self**.

See also: – **initWithObjects:**, – **initWithObjects:count:**, **– initWithSet:, – initWithSet:copyItems:**, +**setWithArray:**

initWithObjects:

– (id)**initWithObjects:**(id)*anObject...*

Initializes a newly allocated set with members taken from the specified list of *objects*. **initWithObjects:** takes a comma-separated list of objects terminated by **nil**. Each object receives a **retain** message as it is added to the set. Returns **self**.

See also: – **initWithArray:**, – **initWithObjects:count:**, **– initWithSet:, – initWithSet:copyItems:**, +**setWithObjects:**

initWithObjects:count:

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

Initializes a newly allocated set with *count* members. This method steps through the *objects* array, creating members in the new set as it goes. Each object receives a **retain** message as it is added to the set. Returns **self**.

See also: – **initWithArray:**, – **initWithObjects:**, **– initWithSet:, – initWithSet:copyItems:**

initWithSet:

– (id)**initWithSet:**(NSSet *)*otherSet*

Initializes a newly allocated set by placing in it the objects contained in *otherSet*. Each object is retained as it is added to the receiver. Returns **self**.

See also: – **initWithArray:**, – **initWithObjects:**, – **initWithObjects:count:**, **– initWithSet:copyItems:**

initWithSet:copyItems:

– (id)**initWithSet:**(NSSet *)*otherSet* **copyItems:**(BOOL)*flag*

Initializes a newly allocated set and, if *flag* is NO, places in it the objects contained in *otherSet*. If *flag* is YES, the members of *otherSet* are copied, and the copies are added to the receiver. (Note that **copyWithZone:** is invoked in making these copies. Thus, the receiver's new member objects may be immutable, even though their counterparts in *otherSet* were mutable. Also, members must conform to the NSCopying protocol)

This method returns **self**.

See also: – **initWithArray:**, – **initWithObjects:**, – **initWithObjects:count:**, **– initWithSet:**

intersectsSet:

– (BOOL)**intersectsSet:**(NSSet *)*otherSet*

Returns YES if at least one object in the receiver is also present in *otherSet*, NO otherwise.

See also: – **isEqualToSet:**, – **isSubsetOfSet:**

isEqual:

@protocol NSObject
– (BOOL)**isEqual:**(id)*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 NSSet and contain the same contents (as determined by the **isEqualToSet:** method).

See also: – **isEqualToSet:**

isEqualToSet:

– (BOOL)**isEqualToSet:**(NSSet *)*otherSet*

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

Two sets have equal contents if they each have the same number of members and if each member of one set is present in the other.

See also: – **intersectsSet:**, – **isEqual:** (NSObject protocol), – **isSubsetOfSet:**

isSubsetOfSet:

– (BOOL)**isSubsetOfSet:**(NSSet *)*otherSet*

Returns YES if every object in the receiver is also present in *otherSet*, NO otherwise.

See also: – **intersectsSet:**, – **isEqualToSet:**

makeObjectsPerform:

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

Sends *aSelector* to each object in the set. The *aSelector* method must be one that takes no arguments. It shouldn't have the side effect of modifying this set. The messages are sent using the **performSelector:** method declared in the NSObject protocol.

See also: – **makeObjectsPerform:withObject:**

makeObjectsPerformSelector:

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

Same as **makeObjectsPerform:**.

makeObjectsPerformSelector:withObject:

– (void)**makeObjectsPerformSelector:**(SEL)*aSelector* **withObject:**(id)*anObject*

Same as **makeObjectsPerform:withObject:**.

makeObjectsPerform:withObject:

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

Sends *aSelector* to each object in the set. The message is sent each time with *anObject* as the 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 this set. The messages are sent using the **performSelector:withObject:** method declared in the NSObject protocol.

See also: – **makeObjectsPerform:**

member:

– (id)**member:**(id)*anObject*

If *anObject* is present in the set (as determined by **isEqual:**), the object in the set is returned. Otherwise, **member:** returns **nil**.

See also: – **containsObject:**

objectEnumerator

– (NSEnumerator *)**objectEnumerator**

Returns an enumerator object that lets you access each object in the set:

```
NSEnumerator *enumerator = [mySet objectEnumerator];
id value;

while ((value = [enumerator nextObject])) {
    /* code that acts on the set's values */
}
```

When this method is used with mutable subclasses of NSSet, your code shouldn't modify the set during enumeration. If you intend to modify the set, use the **allObjects** method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

See also: – **nextObject** (NSEnumerator)

NSMutableSet

Inherits From: NSSet : NSObject

Conforms To: NSCoding
NSCopying
NSMutableCopying (NSSet)
NSObject (NSObject)

Declared In: Foundation/NSSet.h

Class at a Glance

Purpose
An NSMutableSet object stores a modifiable set of objects.

Principal Attributes
- The objects that make up the set.

Creation

+ setWithCapacity:	Returns An empty set with enough allocated memory to hold a specified number of objects.

Commonly Used Methods

– addObject:	Adds an object to the set, if it isn't already a member.
– removeObject:	Removes an object from the set.

Primitive Methods
– addObject:
– removeObject:

Class Description

The NSMutableSet class declares the programmatic interface to an object that manages a mutable set of objects. NSMutableSet provides support for the mathematical concept of a *set*. A set, both in its mathematical sense, and in the NSMutableSet implementation, is an *unordered* collection of distinct

elements. The NSCountedSet class, which is a concrete subclass of NSMutableSet, supports mutable sets that can contain multiple instances of the same element. The NSSet class supports creating and managing immutable sets.

Objects are added to an NSMutableSet with **addObject:**, which adds a single object to the set; **addObjectsFromArray:**, which adds all objects from a specified array to the set; or with **unionSet:**, which adds all the objects from another set.

Objects are removed from an NSMutableSet using any of the methods **intersectSet:**, **minusSet:**, **removeAllObjects**, or **removeObject:**.

Method Types

Creating an NSMutableSet	+ allocWithZone: + setWithCapacity: – initWithCapacity:
Adding and removing entries	– addObject: – removeObject: – removeAllObjects – addObjectsFromArray:
Combining and recombining sets	– unionSet: – minusSet: – intersectSet: – setSet:

Class Methods

allocWithZone:

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

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

Typically, you create temporary sets using the **set...** class methods, not the **allocWithZone:** and **init...** methods.

See also: + **initWithCapacity:**, + **set** (NSSet), + **setWithObjects:count:** (NSSet)

setWithCapacity:

+ (id)**setWithCapacity:**(unsigned)*numItems*

Creates and returns a mutable set, giving it enough allocated memory to hold *numItems* members. Mutable sets allocate additional memory as needed, so *numItems* simply establishes the object's initial capacity.

See also: – **initWithCapacity:**, + **set** (NSSet), + **setWithObjects:count:** (NSSet)

nstance Methods

addObject:

– (void)**addObject:**(id)*anObject*

Adds the specified object to the receiver if it is not already a member. *anObject* is sent a **retain** message as it is added to the receiver. If *anObject* is already present in the set, this method has no effect on either the set or on *anObject*.

See also: – **addObjectsFromArray:**, – **unionSet:**

addObjectsFromArray:

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

Adds each object contained in *anArray* to the receiver, if that object is not already a member. The new member is retained. If a given element of the array is already present in the set, this method has no effect on either the set or on the array element.

See also: – **addObject:**, – **unionSet:**

initWithCapacity:

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

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

See also: + **setWithCapacity:**

intersectSet:

– (void)**intersectSet:**(NSSet *)*otherSet*

Removes from the receiver each object that isn't a member of *otherSet*. Each object that's removed from the receiver is sent a **release** message.

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

minusSet:

– (void)**minusSet:**(NSSet *)*otherSet*

Removes from the receiver each object contained in *otherSet* that is also present in the receiver. Each object that's successfully removed from the receiver is sent a **release** message. If any member of *otherSet* isn't present in the receiving set, this method has no effect on either the receiver or on the *otherSet* member.

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

removeAllObjects

– (void)**removeAllObjects**

Empties the set of all of its members. Each member is sent a **release** message.

See also: – **removeObject:**, – **minusSet:**, – **intersectSet:**

removeObject:

– (void)**removeObject:**(id)*anObject*

Removes *anObject* from the set. The removed object is sent a **release** message if it was a member of the receiver.

See also: – **removeAllObjects**, – **minusSet:**, – **intersectSet:**

setSet:

– (void)**setSet:**(NSSet *)*otherSet*

Empties the receiver, then adds each object contained in *otherSet* to the receiver The new member is sent a **retain** message as it is added to the receiver.

unionSet:

– (void)**unionSet:**(NSSet *)*otherSet*

Adds each object contained in *otherSet* to the receiver, if that object is not already a member. The new member is sent a **retain** message as it is added to the receiver. If any member of *otherSet* is already present in the receiver, this method has no effect on either the receiver or on the *otherSet* member.

See also: – **addObject:**, – **addObjectsFromArray:**