NSSet Class Reference

Cocoa > Data Management

2008-10-15

Contents

NSSet Class Reference 5

Overview 5
Adopted Protocols 6
Tasks 7
- Creating a Set 7
- Initializing a Set 7
- Counting Entries 8
- Accessing Set Members 8
- Comparing Sets 8
- Key-Value Observing 9
- Describing a Set 9

Class Methods 9
- set 9
- setWithArray: 9
- setWithObject: 10
- setWithObjects: 11
- setWithObjects:count: 11
- setWithSet: 12

Instance Methods 13
- addObserver:forKeyPath:options:context: 13
- allObjects 13
- anyObject 14
- containsObject: 14
- count 15
- description 15
- descriptionWithLocale: 15
- filteredSetUsingPredicate: 16
- initWithArray: 17
- initWithObjects: 17
- initWithObjects:count: 18
- initWithSet: 18
- initWithSet:copyItems: 19
- intersectsSet: 20
- isEqualToSet: 20
- isSubsetOfSet: 21
- makeObjectsPerformSelector: 21
- makeObjectsPerformSelector:withObject: 21
- member: 22
- objectEnumerator 22
- removeObserver:forKeyPath: 23
- setByAddingObject: 24

setByAddingObjectsFromArray: 24
setByAddingObjectsFromSet: 25
setValue:forKey: 25
valueForKey: 26

Document Revision History 27

Index 29

NSSet Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSFastEnumeration NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.0 and later.
Companion guide	Collections Programming Topics for Cocoa
Declared in	NSKeyValueCoding.h NSKeyValueObserving.h NSPredicate.h NSSet.h
Related sample code	Core Data HTML Store CoreRecipes CustomAtomicStoreSubclass NewsReader Sketch-112

Overview

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

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

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

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

Objects in a set must respond to the `NSObject` protocol methods `hash` and `isEqual:`—see the `NSObject` protocol for more information.

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

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

Typically, you create a temporary set by sending one of the `set`… methods to the `NSSet` class object. These methods return an `NSSet` object containing the elements (if any) you pass in as arguments. The `set` (page 9) method is a "convenience" method to create an empty mutable set.

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

`NSSet` provides methods for querying the elements of the set. `allObjects` (page 13) returns an array containing the objects in a set. `anyObject` (page 14) returns some object in the set. `count` (page 15) returns the number of objects currently in the set. `member:` (page 22) returns the object in the set that is equal to a specified object. Additionally, `intersectsSet:` (page 20) tests for set intersection, `isEqualToSet:` (page 20) tests for set equality, and `isSubsetOfSet:` (page 21) tests for one set being a subset of another.

The `objectEnumerator` (page 22) method provides for traversing elements of the set one by one. For better performance on Mac OS X v10.5 and later, you can also use the Objective-C fast enumeration feature (see Fast Enumeration).

`NSSet`'s `makeObjectsPerformSelector:` (page 21) and `makeObjectsPerformSelector:withObject:` (page 21) methods provides for sending messages to individual objects in the set.

`NSSet` is "toll-free bridged" with its Core Foundation counterpart, *CFSet Reference*. This means that the Core Foundation type is interchangeable in function or method calls with the bridged Foundation object. Therefore, in a method where you see an `NSSet *` parameter, you can pass a `CFSetRef`, and in a function where you see a `CFSetRef` parameter, you can pass an `NSSet` instance (you cast one type to the other to suppress compiler warnings). See Interchangeable Data Types for more information on toll-free bridging.

Adopted Protocols

NSCoding

`encodeWithCoder:`

`initWithCoder:`

NSCopying
`copyWithZone:`

NSMutableCopying
`mutableCopyWithZone:`

Tasks

Creating a Set

`+ set` (page 9)
Creates and returns an empty set.

`+ setWithArray:` (page 9)
Creates and returns a set containing a uniqued collection of those objects contained in a given array.

`+ setWithObject:` (page 10)
Creates and returns a set that contains a single given object.

`+ setWithObjects:` (page 11)
Creates and returns a set containing the objects in a given argument list.

`+ setWithObjects:count:` (page 11)
Creates and returns a set containing a specified number of objects from a given C array of objects.

`+ setWithSet:` (page 12)
Creates and returns a set containing the objects from another set.

`- setByAddingObject:` (page 24)
Returns a new set formed by adding a given object to the collection defined by the receiver.

`- setByAddingObjectsFromSet:` (page 25)
Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

`- setByAddingObjectsFromArray:` (page 24)
Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

Initializing a Set

`- initWithArray:` (page 17)
Initializes a newly allocated set with the objects that are contained in a given array.

`- initWithObjects:` (page 17)
Initializes a newly allocated set with members taken from the specified list of objects.

`- initWithObjects:count:` (page 18)
Initializes a newly allocated set with a specified number of objects from a given C array of objects.

`- initWithSet:` (page 18)
Initializes a newly allocated set and adds to it objects from another given set.

`- initWithSet:copyItems:` (page 19)
Initializes a newly allocated set and adds to it members of another given set.

Counting Entries

`– count` (page 15)
: Returns the number of members in the receiver.

Accessing Set Members

`– allObjects` (page 13)
: Returns an array containing the receiver's members, or an empty array if the receiver has no members.

`– anyObject` (page 14)
: Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.

`– containsObject:` (page 14)
: Returns a Boolean value that indicates whether a given object is present in the receiver.

`– filteredSetUsingPredicate:` (page 16)
: Evaluates a given predicate against each object in the receiver and returns a new set containing the objects for which the predicate returns true.

`– makeObjectsPerformSelector:` (page 21)
: Sends to each object in the receiver a message specified by a given selector.

`– makeObjectsPerformSelector:withObject:` (page 21)
: Sends to each object in the receiver a message specified by a given selector.

`– member:` (page 22)
: Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.

`– objectEnumerator` (page 22)
: Returns an enumerator object that lets you access each object in the receiver.

Comparing Sets

`– isSubsetOfSet:` (page 21)
: Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.

`– intersectsSet:` (page 20)
: Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.

`– isEqualToSet:` (page 20)
: Compares the receiver to another set.

`– valueForKey:` (page 26)
: Return a set containing the results of invoking `valueForKey:` on each of the receiver's members.

`– setValue:forKey:` (page 25)
: Invokes `setValue:forKey:` on each of the receiver's members.

Key-Value Observing

`- addObserver:forKeyPath:options:context:` (page 13)
  Raises an exception.

`- removeObserver:forKeyPath:` (page 23)
  Raises an exception.

Describing a Set

`- description` (page 15)
  Returns a string that represents the contents of the receiver, formatted as a property list.

`- descriptionWithLocale:` (page 15)
  Returns a string that represents the contents of the receiver, formatted as a property list.

Class Methods

set

Creates and returns an empty set.

```
+ (id)set
```

Return Value
A new empty set.

Discussion
This method is declared primarily for the use of mutable subclasses of `NSSet`.

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

See Also
`+ setWithArray:` (page 9)
`+ setWithObject:` (page 10)
`+ setWithObjects:` (page 11)
`- setByAddingObject:` (page 24)
`- setByAddingObjectsFromSet:` (page 25)
`- setByAddingObjectsFromArray:` (page 24)

Declared In
`NSSet.h`

setWithArray:

Creates and returns a set containing a uniqued collection of those objects contained in a given array.

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

Parameters

anArray

An array containing the objects to add to the new set. If the same object appears more than once in *anArray*, it is added only once to the returned set. Each object receives a `retain` message as it is added to the set.

Return Value

A new set containing a uniqued collection of those objects contained in *anArray*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ set` (page 9)
`+ setWithObject:` (page 10)
`+ setWithObjects:` (page 11)
`- setByAddingObject:` (page 24)
`- setByAddingObjectsFromSet:` (page 25)
`- setByAddingObjectsFromArray:` (page 24)

Related Sample Code

CoreRecipes

NewsReader

Declared In

`NSSet.h`

setWithObject:

Creates and returns a set that contains a single given object.

```
+ (id)setWithObject:(id)anObject
```

Parameters

anObject

The object to add to the new set. *anObject* receives a `retain` message after being added to the set.

Return Value

A new set that contains a single member, *anObject*.

Availability

Available in Mac OS X v10.0 and later.

See Also

`+ set` (page 9)
`+ setWithArray:` (page 9)
`+ setWithObjects:` (page 11)
`- setByAddingObject:` (page 24)
`- setByAddingObjectsFromSet:` (page 25)
`- setByAddingObjectsFromArray:` (page 24)

Related Sample Code
Core Data HTML Store

Declared In
`NSSet.h`

setWithObjects:

Creates and returns a set containing the objects in a given argument list.

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

Parameters

anObject
The first object to add to the new set.

anObject, ...
A comma-separated list of objects, ending with `nil`, to add to the new set. If the same object appears more than once in the list of objects, it is added only once to the returned set. Each object receives a `retain` message as it is added to the set.

Return Value
A new set containing the objects in the argument list.

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

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

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

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

See Also
+ `set` (page 9)
+ `setWithArray:` (page 9)
+ `setWithObject:` (page 10)
- `setByAddingObject:` (page 24)
- `setByAddingObjectsFromSet:` (page 25)
- `setByAddingObjectsFromArray:` (page 24)

Declared In
`NSSet.h`

setWithObjects:count:

Creates and returns a set containing a specified number of objects from a given C array of objects.

```
+ (id)setWithObjects:(id *)objects  count:(NSUInteger)count
```

Parameters

objects

A C array of objects to add to the new set. If the same object appears more than once in *objects*, it is added only once to the returned set. Each object receives a `retain` message as it is added to the set.

count

The number of objects from *objects* to add to the new set.

Return Value

A new set containing *count* objects from the list of objects specified by *objects*.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ `set` (page 9)
+ `setWithArray:` (page 9)
+ `setWithObject:` (page 10)
+ `setWithObjects:` (page 11)
- `setByAddingObject:` (page 24)
- `setByAddingObjectsFromSet:` (page 25)
- `setByAddingObjectsFromArray:` (page 24)

Declared In

`NSSet.h`

setWithSet:

Creates and returns a set containing the objects from another set.

```
+ (id)setWithSet:(NSSet *)aSet
```

Parameters

aSet

A set containing the objects to add to the new set. Each object receives a `retain` message as it is added to the new set.

Return Value

A new set containing the objects from *aSet*.

Availability

Available in Mac OS X v10.0 and later.

See Also

+ `set` (page 9)
+ `setWithArray:` (page 9)
+ `setWithObject:` (page 10)
+ `setWithObjects:` (page 11)
- `setByAddingObject:` (page 24)
- `setByAddingObjectsFromSet:` (page 25)

`- setByAddingObjectsFromArray:` (page 24)

Declared In
`NSSet.h`

Instance Methods

addObserver:forKeyPath:options:context:

Raises an exception.

```
- (void)addObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath
    options:(NSKeyValueObservingOptions)options context:(void *)context
```

Parameters

observer
: The object to register for KVO notifications. The observer must implement the key-value observing method `observeValueForKeyPath:ofObject:change:context:`.

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

options
: A combination of the `NSKeyValueObservingOptions` values that specifies what is included in observation notifications. For possible values, see `NSKeyValueObservingOptions`.

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

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

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

See Also
`- removeObserver:forKeyPath:` (page 23)

Declared In
`NSKeyValueObserving.h`

allObjects

Returns an array containing the receiver's members, or an empty array if the receiver has no members.

```
- (NSArray *)allObjects
```

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

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

See Also
`– anyObject` (page 14)
`– objectEnumerator` (page 22)

Related Sample Code
CoreRecipes
Sketch-112

Declared In
`NSSet.h`

anyObject

Returns one of the objects in the receiver, or `nil` if the receiver contains no objects.

```
- (id)anyObject
```

Return Value
One of the objects in the receiver, or `nil` if the receiver contains no objects. The object returned is chosen at the receiver's convenience—the selection is not guaranteed to be random.

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

See Also
`– allObjects` (page 13)
`– objectEnumerator` (page 22)

Related Sample Code
Core Data HTML Store

Declared In
`NSSet.h`

containsObject:

Returns a Boolean value that indicates whether a given object is present in the receiver.

```
- (BOOL)containsObject:(id)anObject
```

Parameters

`anObject`
The object for which to test membership of the receiver.

Return Value
`YES` if *`anObject`* is present in the receiver, otherwise `NO`.

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

See Also
`- member:` (page 22)

Declared In
`NSSet.h`

count

Returns the number of members in the receiver.

```
- (NSUInteger)count
```

Return Value
The number of members in the receiver.

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

Related Sample Code
CoreRecipes

Declared In
`NSSet.h`

description

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

```
- (NSString *)description
```

Return Value
A string that represents the contents of the receiver, formatted as a property list.

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

See Also
`- descriptionWithLocale:` (page 15)

Declared In
`NSSet.h`

descriptionWithLocale:

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

```
- (NSString *)descriptionWithLocale:(id)locale
```

Parameters

locale

In Mac OS X v10.4 and earlier, this must be a dictionary that specifies options used for formatting each of the receiver's members. In Mac OS X v10.5 and later, you can use an `NSLocale` object. If you do not want the receiver's members to be formatted, specify `nil`.

Return Value

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

Discussion

This method sends each of the receiver's members `descriptionWithLocale:` with *locale* passed as the sole parameter. If the receiver's members do not respond to `descriptionWithLocale:`, this method sends `description` instead.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- description` (page 15)

Declared In

`NSSet.h`

filteredSetUsingPredicate:

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

```
- (NSSet *)filteredSetUsingPredicate:(NSPredicate *)predicate
```

Parameters

predicate

A predicate.

Return Value

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

Discussion

The following example illustrates the use of this method.

```
NSSet *sourceSet =
    [NSSet setWithObjects:@"One", @"Two", @"Three", @"Four", nil];
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith 'T'"];
NSSet *filteredSet =
    [sourceSet filteredSetUsingPredicate:predicate];
// filteredSet contains (Two, Three)
```

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSPredicate.h`

initWithArray:

Initializes a newly allocated set with the objects that are contained in a given array.

```
- (id)initWithArray:(NSArray *)array
```

Parameters

array

An array of objects to add to the new set. If the same object appears more than once in *array*, it is represented only once in the returned set. Each object receives a `retain` message as it is added to the set.

Return Value

An initialized object, which might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithObjects:` (page 17)
- `initWithObjects:count:` (page 18)
- `initWithSet:` (page 18)
- `initWithSet:copyItems:` (page 19)

+ `setWithArray:` (page 9)

Declared In

`NSSet.h`

initWithObjects:

Initializes a newly allocated set with members taken from the specified list of objects.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters

anObject

The first object to add to the new set.

firstObj, ...

A comma-separated list of objects, ending with `nil`, to add to the new set. If the same object appears more than once in the list, it is represented only once in the returned set. Each object receives a `retain` message as it is added to the set

Return Value

An initialized object, which might be different than the original receiver.

Availability

Available in Mac OS X v10.0 and later.

See Also

- `initWithArray:` (page 17)
- `initWithObjects:count:` (page 18)
- `initWithSet:` (page 18)
- `initWithSet:copyItems:` (page 19)

`+ setWithObjects:` (page 11)

Declared In
`NSSet.h`

initWithObjects:count:

Initializes a newly allocated set with a specified number of objects from a given C array of objects.

```
- (id)initWithObjects:(id *)objects  count:(NSUInteger)count
```

Parameters

`objects`
A C array of objects to add to the new set. If the same object appears more than once in *`objects`*, it is added only once to the returned set. Each object receives a `retain` message as it is added to the set.

`count`
The number of objects from *`objects`* to add to the new set.

Return Value
An initialized object, which might be different than the original receiver.

Discussion
This method is the designated initializer for `NSSet`.

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

See Also
`- initWithArray:` (page 17)
`- initWithObjects:` (page 17)
`- initWithSet:` (page 18)
`- initWithSet:copyItems:` (page 19)
`+ setWithObjects:count:` (page 11)

Declared In
`NSSet.h`

initWithSet:

Initializes a newly allocated set and adds to it objects from another given set.

```
- (id)initWithSet:(NSSet *)otherSet
```

Parameters

`otherSet`
A set containing objects to add to the receiver. Each object is retained as it is added to the receiver.

Return Value
An initialized object, which might be different than the original receiver.

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

See Also
- `- initWithArray:` (page 17)
- `- initWithObjects:` (page 17)
- `- initWithObjects:count:` (page 18)
- `- initWithSet:copyItems:` (page 19)
- `+ setWithSet:` (page 12)

Declared In
`NSSet.h`

initWithSet:copyItems:

Initializes a newly allocated set and adds to it members of another given set.

```
- (id)initWithSet:(NSSet *)otherSet copyItems:(BOOL)flag
```

Parameters

otherSet
A set containing objects to add to the new set.

flag
If `YES`, the members of *otherSet* are copied, and the copies are added to the receiver. If `NO`, the members of *otherSet* are added to the receiver and retained.

Return Value
An initialized object that contains the members of *otherSet*.

This method returns an initialized object, which might be different than the original receiver.

Discussion
Note that, if *flag* is `YES`, `copyWithZone:` is invoked to make copies—thus, the receiver's new member objects may be immutable, even though their counterparts in *otherSet* were mutable. Also, members must conform to the `NSCopying` protocol)

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

See Also
- `- initWithArray:` (page 17)
- `- initWithObjects:` (page 17)
- `- initWithObjects:count:` (page 18)
- `- initWithSet:` (page 18)
- `+ setWithSet:` (page 12)

Declared In
`NSSet.h`

intersectsSet:

Returns a Boolean value that indicates whether at least one object in the receiver is also present in another given set.

```
- (BOOL)intersectsSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to compare the receiver.

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

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

See Also
`- isEqualToSet:` (page 20)
`- isSubsetOfSet:` (page 21)

Declared In
`NSSet.h`

isEqualToSet:

Compares the receiver to another set.

```
- (BOOL)isEqualToSet:(NSSet *)otherSet
```

Parameters

otherSet
The set with which to compare the receiver.

Return Value
`YES` if the contents of *otherSet* are equal to the contents of the receiver, otherwise `NO`.

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

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

See Also
`- intersectsSet:` (page 20)
`- isEqual:` (`NSObject` protocol)
`- isSubsetOfSet:` (page 21)

Declared In
`NSSet.h`

isSubsetOfSet:

Returns a Boolean value that indicates whether every object in the receiver is also present in another given set.

```
- (BOOL)isSubsetOfSet:(NSSet *)otherSet
```

Parameters

otherSet

The set with which to compare the receiver.

Return Value

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

Availability

Available in Mac OS X v10.0 and later.

See Also

`- intersectsSet:` (page 20)

`- isEqualToSet:` (page 20)

Declared In

`NSSet.h`

makeObjectsPerformSelector:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector
```

Parameters

aSelector

A selector that specifies the message to send to the members of the receiver. The method must not take any arguments. It should not have the side effect of modifying the receiver. This value must not be `NULL`.

Discussion

The message specified by *aSelector* is sent once to each member of the receiver. This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- makeObjectsPerformSelector:withObject:` (page 21)

Declared In

`NSSet.h`

makeObjectsPerformSelector:withObject:

Sends to each object in the receiver a message specified by a given selector.

```
- (void)makeObjectsPerformSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector

A selector that specifies the message to send to the receiver's members. The method must take a single argument of type `id`. The method should not, as a side effect, modify the receiver. The value must not be `NULL`.

anObject

The object to pass as an argument to the method specified by *aSelector*.

Discussion

The message specified by *aSelector* is sent, with *anObject* as the argument, once to each member of the receiver. This method raises an `NSInvalidArgumentException` if *aSelector* is `NULL`.

Availability

Available in Mac OS X v10.0 and later.

See Also

`- makeObjectsPerformSelector:` (page 21)

Declared In

`NSSet.h`

member:

Determines whether the receiver contains an object equal to a given object, and returns that object if it is present.

```
- (id)member:(id)anObject
```

Parameters

anObject

The object for which to test for membership of the receiver.

Return Value

If the receiver contains an object equal to *anObject* (as determined by `isEqual:`) then that object (typically this will be *anObject*), otherwise `nil`.

Discussion

If you override `isEqual:`, you must also override the `hash` method for the `member:` method to work on a set of objects of your class.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSSet.h`

objectEnumerator

Returns an enumerator object that lets you access each object in the receiver.

```
- (NSEnumerator *)objectEnumerator
```

Return Value
An enumerator object that lets you access each object in the receiver.

Discussion
The following code fragment illustrates how you can use this method.

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

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

When this method is used with mutable subclasses of `NSSet`, your code shouldn’t modify the receiver during enumeration. If you intend to modify the receiver, use the `allObjects` (page 13) method to create a “snapshot” of the set’s members. Enumerate the snapshot, but make your modifications to the original set.

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

See Also
`- nextObject` (`NSEnumerator`)

Related Sample Code
CoreRecipes

Declared In
`NSSet.h`

removeObserver:forKeyPath:

Raises an exception.

`- (void)removeObserver:(NSObject *)observer forKeyPath:(NSString *)keyPath`

Parameters

`observer`
The object to remove as an observer.

`keyPath`
A key-path, relative to the receiver, for which *`observer`* is registered to receive KVO change notifications. This value must not be `nil`.

Special Considerations

`NSSet` objects are not observable, so this method raises an exception when invoked on an `NSSet` object. Instead of observing a set, observe the unordered to-many relationship for which the set is the collection of related objects.

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

See Also
`- addObserver:forKeyPath:options:context:` (page 13)

Declared In
`NSKeyValueObserving.h`

setByAddingObject:

Returns a new set formed by adding a given object to the collection defined by the receiver.

```
- (NSSet *)setByAddingObject:(id)anObject
```

Parameters

anObject
The object to add to the collection defined by the receiver.

Return Value
A new set formed by adding *anObject* to the collection defined by the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ set` (page 9)
`+ setWithArray:` (page 9)
`+ setWithObject:` (page 10)
`+ setWithObjects:` (page 11)
`- setByAddingObjectsFromSet:` (page 25)
`- setByAddingObjectsFromArray:` (page 24)

Declared In
`NSSet.h`

setByAddingObjectsFromArray:

Returns a new set formed by adding the objects in a given array to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromArray:(NSArray *)other
```

Parameters

other
The array of objects to add to the collection defined by the receiver.

Return Value
A new set formed by adding the objects in *other* to the collection defined by the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ set` (page 9)
`+ setWithArray:` (page 9)
`+ setWithObject:` (page 10)
`+ setWithObjects:` (page 11)
`- setByAddingObject:` (page 24)

`- setByAddingObjectsFromSet:` (page 25)

Declared In
`NSSet.h`

setByAddingObjectsFromSet:

Returns a new set formed by adding the objects in a given set to the collection defined by the receiver.

```
- (NSSet *)setByAddingObjectsFromSet:(NSSet *)other
```

Parameters

`other`
The set of objects to add to the collection defined by the receiver.

Return Value
A new set formed by adding the objects in *`other`* to the collection defined by the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ set` (page 9)
`+ setWithArray:` (page 9)
`+ setWithObject:` (page 10)
`+ setWithObjects:` (page 11)
`- setByAddingObject:` (page 24)
`- setByAddingObjectsFromSet:` (page 25)

Declared In
`NSSet.h`

setValue:forKey:

Invokes `setValue:forKey:` on each of the receiver's members.

```
- (void)setValue:(id)value forKey:(NSString *)key
```

Parameters

`value`
The value for the property identified by *`key`*.

`key`
The name of one of the properties of the receiver's members.

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

See Also
`- valueForKey:` (page 26)

Declared In
`NSKeyValueCoding.h`

valueForKey:

Return a set containing the results of invoking `valueForKey:` on each of the receiver's members.

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

Parameters

key

The name of one of the properties of the receiver's members.

Return Value

A set containing the results of invoking `valueForKey:` (with the argument *key*) on each of the receiver's members.

Discussion

The returned set might not have the same number of members as the receiver. The returned set will not contain any elements corresponding to instances of `valueForKey:` returning `nil` (note that this is in contrast with `NSArray`'s implementation, which may put `NSNull` values in the arrays it returns).

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setValue:forKey:` (page 25)

Declared In

`NSKeyValueCoding.h`

Document Revision History

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

Date	Notes
2008-10-15	Corrected typographical errors.
2008-03-11	Added NSFastEnumeration to list of adopted protocols.
2008-02-08	Corrected the definition of the member: method.
2007-12-11	Corrected typographical error.
2007-04-02	Included API introduced in Mac OS X v10.5.
2006-05-23	First publication of this content as a separate document.

Index

A

addObserver:forKeyPath:options:context: instance method 13
allObjects instance method 13
anyObject instance method 14

C

containsObject: instance method 14
count instance method 15

D

description instance method 15
descriptionWithLocale: instance method 15

F

filteredSetUsingPredicate: instance method 16

I

initWithArray: instance method 17
initWithObjects: instance method 17
initWithObjects:count: instance method 18
initWithSet: instance method 18
initWithSet:copyItems: instance method 19
intersectsSet: instance method 20
isEqualToSet: instance method 20
isSubsetOfSet: instance method 21

M

makeObjectsPerformSelector: instance method 21
makeObjectsPerformSelector:withObject: instance method 21
member: instance method 22

O

objectEnumerator instance method 22

R

removeObserver:forKeyPath: instance method 23

S

set class method 9
setByAddingObject: instance method 24
setByAddingObjectsFromArray: instance method 24
setByAddingObjectsFromSet: instance method 25
setValue:forKey: instance method 25
setWithArray: class method 9
setWithObject: class method 10
setWithObjects: class method 11
setWithObjects:count: class method 11
setWithSet: class method 12

V

valueForKey: instance method 26