MiscSortedCollection Class Cluster

Class Cluster Description

MiscSortedCollection objects manage sorted collections of objects. The cluster's two public classes, MiscSortedCollection and MiscMutableSortedCollection, declare the programmatic interface for static and dynamic collections, respectively.

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

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

The MiscSortedCollection class adopts the NSCopying and NSMutableCopying protocols, making it convenient to convert a collection of one type to the other.

Objects in a MiscSortedCollection are sorted NSOrderedAscending. Order is determined internally through object comparison. Duplicates are allowed (more than one object NSOrderedSame). The collections support insertion, deletion and finding in O(log n) time.

NSNumbers and NSStrings already have comparison methods. If you instantiate a NSNumber collection or a NSString collection you can only insert NSNumbers or respectively NSStrings in the collection. For collections with NSObjects you

have to specify the comparison method or function.

The MiscSortedCollection class cluster uses skiplists internally. SkipLists are data structures that use probalistic balancing rather than strictly enforcing balancing. As a result, the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than algorithms for balanced trees.

Reading material about skiplist:

· Pugh W., Skip Lists: A Probalistic Alternative to Balanced Tree
Communications of the ACM, June 1990, Volume 33, Number 6, p. 668-676
· Pugh W., A skip list cookbook
Tech. Rep. CS-TR-2286.1, Dept.of Computer Science, Univ. of Maryland, College Park,MD [July 1989]

MiscSortedCollection

Inherits From: NSObject

Conforms To: NSCopying
NSMutableCopying
NSObject (NSObject)

Declared In: MiscSortedCollection.h

Class Description

The MiscSortedCollection class declares the programmatic interface to an object that manages an immutable sorted collection of objects. MiscSortedCollection's five primitive methodsÐ**count**, **objectBefore:**, **objectEnumerator**, **smallest** and **allObjectsOrderedSameAs:**Ðprovide the basis for all the other methods in its interface. The **count** method returns the number of elements in the collection. **objectBefore:** returns the largest object which compares NSOrderedDescending to the argument. **objectEnumerator** permits sequential access of the elements of the collection. **smallest** returns the smallest object in the collection. **allObjectsOrderedSameAs:** returns an array containing all the objects which compare NSOrderedSame with the argument.

Instance Variables

None declared in this class.

Adopted Protocols

NSCopying	- copy - copyWithZone:
NSMutableCopying	- mutableCopy - mutableCopyWithZone:

Method Types

Allocating and initializing	+ allocWithZone: + collection + collectionWithCompareFunction:context:

	+ collectionWithCompareSelector: + collectionWithNumbers: + collectionWithNumbers:count: + collectionWithObjects: + collectionWithObjects:compareFunction:context: + collectionWithObjects:compareSelector: + collectionWithObjects:count: + collectionWithObjects:count:compareFunction:context: + collectionWithObjects:count:compareSelector: + collectionWithStrings: + collectionWithStrings:count: + collectionWithStrings:count:options: + collectionWithStrings:count:options:range: + collectionWithStrings:options: + collectionWithStrings:options:range: - initWithObjects:count: - initWithObjects:count:compareFunction:context: - initWithObjects:count:compareSelector: + numberCollection + stringCollection + stringCollectionOptions: + stringCollectionOptions:range:
Counting entries	- count
Accessing objects	- allObjects - allObjectsOrderedSameAs: - objectBefore: - objectEnumerator - smallest
Comparing collections	- isEqualToSortedCollection:

Joining string elements - componentsJoinedByString:

Creating a description of the array
- description
- descriptionWithIndent:

Class Methods

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

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

Typically, you create collection objects using the **collection...** class methods, not the **alloc...** and **init...** methods. Note that it's your responsibility to free objects created with the **alloc...** methods.

See also: + **collection... methods**

collection
+ **collection**

Creates and returns an empty collection. This method is declared primarily for the use of mutable subclasses of MiscSortedCollection. Objects in the collection are compared with the method **compare:**.

See also: + **collectionWithCompareFunction:context:**, + **collectionWithCompareSelector:**

collectionWithCompareFunction:context:
+ **collectionWithCompareFunction:**(int(*)*(id, id, void *))comparator*
context:(void *)*context*

Creates and returns an empty collection. Objects in the collection are compared with the *comparator* function. The *context* is passed to the *comparator.*

See also: + **collection,** + **collectionWithCompareSelector:**

collectionWithCompareSelector:
+ **collectionWithCompareSelector:**(SEL)*comparator*

Creates and returns an empty collection. Objects in the collection are compared with the *comparator* method.

See also: + **collection,** + **collectionWithCompareFunction:context:**

collectionWithNumbers:
+ **collectionWithNumbers:**(NSArray *)*numbers*

Creates and returns a collection object containing the NSNumber objects from the *numbers* array.

See also: + **collectionWithNumbers:count:**

collectionWithNumbers:count:
+ **collectionWithNumbers:**(NSNumber **)*numbers*
count:(unsigned)*count*

Creates and returns a collection object containing *count* NSNumber objects from the *numbers* memory buffer.

See also: + **collectionWithNumbers:**

collectionWithObjects:
+ **collectionWithObjects:**(NSArray *)*objects*

Creates and returns a collection object containing the objects from the *objects* array. Objects in the collection are compared with the method **compare:**.

See also: + **collectionWithObjects:compareFunction:context:**, + **collectionWithObjects:compareSelector:**

collectionWithObjects:compareFunction:context:
+ **collectionWithObjects:**(NSArray *)*objects*
compareFunction:(int(*)*(id, id, void *))comparator*
context:(void *)*context*

Creates and returns a collection object containing the objects from the *objects* array. Objects in the collection are compared with the *comparator* function. The *context* is passed to the *comparator.*

See also: + **collectionWithObjects:**, + **collectionWithObjects:compareSelector:**

collectionWithObjects:compareSelector:
+ **collectionWithObjects:**(NSArray *)*objects*
compareSelector:(SEL)*comparator*

Creates and returns a collection object containing the objects from the *objects* array. Objects in the collection are compared with the *comparator* method.

See also: + **collectionWithObjects:**, + **collectionWithObjects:compareFunction:context:**

collectionWithObjects:count:
+ **collectionWithObjects:**(id *)*objects*
count:(unsigned)*count*

Creates and returns a collection object containing *count* objects from the *objects* memory buffer. Objects in the collection are compared with the method **compare:**.

See also: + **collectionWithObjects:count:compareFunction:context:**,

+**collectionWithObjects:count:compareSelector:**

collectionWithObjects:count:compareFunction:context:

+ **collectionWithObjects:**(id *)*objects*
  count:(unsigned)*count*
  compareFunction:(int(*)*(id, id, void *))comparator*
  context:(void *)*context*

Creates and returns a collection object containing *count* objects from the *objects* memory buffer. Objects in the collection are compared with the *comparator* function. The *context* is passed to the *comparator.*

See also: + **collectionWithObjects:count:**, +**collectionWithObjects:count:compareSelector:**

collectionWithObjects:count:compareSelector:

+ **collectionWithObjects:**(id *)*objects*
  count:(unsigned)*count*
  compareSelector:(SEL)*comparator*

Creates and returns a collection object containing *count* objects from the *objects* memory buffer. Objects in the collection are compared with the *comparator* method.

See also: + **collectionWithObjects:count:**, + **collectionWithObjects:count:compareFunction:context:**

collectionWithStrings:

+ **collectionWithStrings:**(NSArray *)*strings*

Creates and returns a collection object containing *count* NSString objects from the *strings* array.

See also: +**collectionWithStrings:options:**, +**collectionWithStrings:options:range:**

collectionWithStrings:count:

+ **collectionWithStrings:**(NSString **)*strings*
  count:(unsigned)*count*

Creates and returns a collection object containing *count* NSString objects from the *strings* memory buffer.

See also: +**collectionWithStrings:count:options:**, +**collectionWithStrings:count:options:range:**

collectionWithStrings:count:options:

+ **collectionWithStrings:**(NSString **)*strings*
  count:(unsigned)*count*
  options:(unsigned)*mask*

Creates and returns a collection object containing *count* NSString objects from the *strings* memory buffer. *mask* is passed to the NSString comparison method.

See also: +**collectionWithStrings:count:options:range:**

collectionWithStrings:count:options:range:

+ **collectionWithStrings:**(NSString **)*strings*
  count:(unsigned)*count*
  options:(unsigned)*mask*
  range:(NSRange)*aRange*

Creates and returns a collection object containing *count* NSString objects from the *strings* memory buffer. *mask* and *aRange* are passed to the NSString comparison method.

See also: +**collectionWithStrings:count:options:**

collectionWithStrings:options:

+ **collectionWithStrings:**(NSArray *)*strings*

options:(unsigned)*mask*

Creates and returns a collection object containing *count* NSString objects from the *strings* array. *mask* is passed to the NSString comparison method.

See also: +**collectionWithStrings:options:range:**

collectionWithStrings:options:range:

+ **collectionWithStrings:**(NSArray *)*strings*
options:(unsigned)*mask*
range:(NSRange)*aRange*

Creates and returns a collection object containing *count* NSString objects from the *strings* array. *mask* and *aRange* are passed to the NSString comparison method.

See also: +**collectionWithStrings:options:**

numberCollection

+ **numberCollection**

Creates and returns an empty NSNumber collection. This method is declared primarily for the use of mutable subclasses of MiscSortedCollection.

See also: + **collectionWithNumbers:**, + **collectionWithNumbers:count:**

stringCollection

+ **stringCollection**

Creates and returns an empty NSString collection. This method is declared primarily for the use of mutable subclasses of MiscSortedCollection.

See also: + **stringCollectionOptions:**, + **stringCollectionOptions:range:**

stringCollectionOptions:

+ **stringCollectionOptions:**(unsigned)*mask*

Creates and returns an empty NSString collection. *mask* is passed to the NSString comparison method.

See also: + **stringCollectionOptions:range:**

stringCollectionOptions:range:

+ **stringCollectionOptions:**(unsigned)*mask*
  range:(NSRange)*aRange*

Creates and returns an empty NSString collection. *mask* and *aRange* are passed to the NSString comparison method.

See also: + **stringCollectionOptions:**

Instance Methods

allObjects

- (NSArray *)**allObjects**

Returns an array containing all the objects in the collection. This snapshots the set of objects. Order is smallest to largest

See also: -**allObjectsOrderedSameAs:**, -**objectEnumerator**

allObjectsOrderedSameAs:

- (NSArray *)**allObjectsOrderedSameAs:***anObject*

Returns an array containing all the objects which compare NSOrderedSame with *anObject*. Order in the array is undefined.

See also: -**allObjects**, -**smallest**

count

- (unsigned)**count**

Returns the number of objects in the collection.

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**

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 collection object, **hash** returns the number of entries in the collection. If two collection objects are equal (as determined by the **isEqual:** method), they will have the same hash value.

See also: - **isEqual:**

initWithObjects:count:

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

Initializes a newly allocated collection object by placing in it *count* objects contained in the *objects* memory buffer. Objects in the collection are compared with the method **compare:**. Returns **self**.

See also: - **initWithObjects:count:compareFunction:context:**, - **initWithObjects:count:compareSelector:**

initWithObjects:count:compareFunction:context:

- **initWithObjects:**(id *)*objects*
  count:(unsigned)*count*
  compareFunction:(int(*)*(id, id, void *))comparator*
  context:(void *)*context*

Initializes a newly allocated collection object by placing in it *count* objects contained in the *objects* memory buffer. Objects in the collection are compared with the function *comparator* in the context *context*. Returns **self**.

See also: - **initWithObjects:count:**, - **initWithObjects:count:compareSelector:**

initWithObjects:count:compareSelector:

- **initWithObjects:**(id *)*objects*
  count:(unsigned)*count*
  compareSelector:(SEL)*comparator*

Initializes a newly allocated collection object by placing in it *count* objects contained in the *objects* memory buffer. Objects in the collection are compared with the method *comparator*. Returns **self**.

See also: - **initWithObjects:count:**, - **initWithObjects:count:compareFunction:context:**

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 MiscSortedCollection and contain the same data (as determined by the **isEqualToSortedCollection:** method).

See also: - **isEqualToSortedCollection:**

isEqualToSortedCollection:

- (BOOL)**isEqualToSortedCollection:**(MiscSortedCollection *)*otherSortedCollection*

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

Two collections have equal contents if they each hold the same number of objects and objects at a given index in each collection satisfy the **isEqual:** test.

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

objectBefore:

- **objectBefore:***anObject*

Returns the largest object which compares NSOrderedDescending to *anObject*.

See also: -**smallest**

objectEnumerator

- (NSEnumerator *)**objectEnumerator**

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

```
id <NSEnumerator> enumerator = [myCollection objectEnumerator];
```

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

When this method is used with mutable subclasses of MiscSortedCollection, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the **allObjects** method to create a ªsnapshotº of the collection's objects. Work from this snapshot to modify the objects.

See also: **- allObjects, - nextObject** (NSEnumerator protocol)

smallest

- **smallest**

Returns the smallest object in collection.

See also: **-objectBefore:**

MiscMutableSortedCollection

Inherits From: MiscSortedCollection

Conforms To: NSCoding
NSCopying
NSMutableCopying

NSObject (NSObject)

Declared In: uwkit/MiscSortedCollection.h

Class Description

The MiscMutableSortedCollection class declares the programmatic interface to objects that manage mutable collections of sorted objects. With its three efficient primitive methodsÐ**insertObject:, removeAllObjectsOrderedSameAs:** and **removeAllObjects**Ðthis class adds modification operations to the basic operations it inherits from MiscSortedCollection.

The other methods declared here operate by invoking one of these primitives. The non-primitive methods provide convenient ways of adding or removing multiple objects at a time.

When an object is removed from a mutable collection, it receives an **release** message. If there are no further references to the object, itÂs 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 collection.

Instance Variables

None declared in this class.

Method Types

- insertObject:
- insertObjectsFromArray:
- removeAllObjects
- removeAllObjectsOrderedSameAs:

Instance Methods

insertObject:

- (void)**insertObject:***anObject*

Inserts *anObject* in collection. If object is nil an NSInvalidArgumentException error is raised. Retain is applied to the object inserted.

See also: -**insertObjectsFromArray:**

insertObjectsFromArray:

- (void)**insertObjectsFromArray:**(NSArray *)*anArray*

Inserts all objects from *anArray.*

See also: - **insertObject:**

removeAllObjects

- (void)**removeAllObjects**

Empties collection. Performs **-release** on each object removed.

removeAllObjectsOrderedSameAs:

- (void)**removeAllObjectsOrderedSameAs:***anObject*

Removes objects which compare NSOrderedSame with *anObject*. Performs **-release** on each object removed