FCSortedCollection

Inherits From: FCOrderedCollection : FCCollection : Object

Declared In: FCSortedCollection.h

Class Description

FCSortedCollection is a subclass of FCOrderedCollection that implements the behavior of a collection which automatically keeps itself sorted.

All collection subclasses which require a sorted collection are subclasses of this class. This includes sorted lists, and sorted sets.

Determining the correct ordering of any two objects in the collection is by default determined by sending one object the - **compare:** message with another object as the argument. The - **compare:** method should then return type **FCCompareType**; FC_COMPARE_EQUAL_TO, FC_COMPARE_GREATER_THAN, or FC_COMPARE_LESS_THAN, depending on whether the receiving object is equal to, greater than, or less than the argument object, respectively.

You can change the comparison method by sending the **setSortSelector:** method, documented

below. If you are writing your own compare method, you might find the six primitive comparison functions (documented in the Foundation Functions section) helpful.

Note that if you modify any of the objects in the collection without removing them and re-adding them, you run the risk of the collection no longer being sorted. In this case, you can call the -**sort** method to regain sorted order.

FCSortedCollection is an abstract superclass. You cannot instantiate it directly; in fact, some of its methods are simply stubs in the superclass and return errors when invoked. Its basic purpose is to provide common methods and an orthogonal interface to its two instantiable subclasses, all of which fully adhere to the interface described here. In the documentation below, the term "collection" refers to any non-abstract subclass of FCSortedCollection.

FCSortedCollection inherits from FCOrderedCollection and FCCollection. The interface documented here only covers the methods that are new or different in FCSortedCollection, but all the methods in FCCollection and FCOrderedCollection will work on an FCSortedCollection as well. Refer to the documentation on those classes to complete the description of an FCSortedCollection.

Instance Variables

Inherited from Object
None declared in this class.

Inherited from FCCollection
id **_fc_contents** ;
Class **_fc_class** ;
SEL **_fc_sortSelector** ;
BOOL **_fc_archiveByReference** ;

Inherited from FCOrderedCollection
None declared in this class.

Declared in FCSortedCollection
None declared in this class.

Method Types

Factory Methods	+alloc +allocFromZone:
Initializing	-initWithSortSelector: -initForClass:withSortSelector:
Accessing the Behavior of the Collection	-isSorted
Changing the Contents	-addObject:
Comparing and Sorting	-sortByCompare: -sortSelector -setSortSelector: -sort

Class Methods

alloc

+ **alloc;**

This method cannot be used to create an FCSortedCollection object. FCSortedCollection is an abstract superclass, you should call **alloc** only on its instantiable subclasses. The method is implemented only to prevent you from using it; if you do use it, it generates an error message.

allocFromZone:

+ **allocFromZone:**(NXZone *)*zone;*

This method cannot be used to create an FCSortedCollection object. FCSortedCollection is an abstract superclass, you should call **allocFromZone:** only on its instantiable subclasses. The method is implemented only to prevent you from using it; if you do use it, it generates an error message.

Instance Methods

addObject:

- **addObject:anObject;**

This method is overridden from its superclasses to add *anObject* into the collection in sorted order. Sorting is accomplished by sending *anObject* the collection's sort selector message. See - **setSortSelector:** for details on sorting and the sort selector.

If *anObject* doesn't respond to the sort selector message, it is added to the end of the collection. Subclasses with unique objects will fail and return **nil** if *anObject* is already present in the collection. If the programmer has set a content class, **addObject:** will fail if *anObject* isn't a kind of that class.

See also: **- removeObject:** (FCCollection)**, - uniqueElements** (FCSortedList, FCSortedSet)**, - setContentClass:** (FCCollection)

initForClass:withSortSelector:

- **initForClass:theClass withSortSelector:**(SEL)*theSelector;*

Initializes the receiver, a new FCCollection object, and sets it to only allow objects that are a kind of *theClass* to be added to its collection. The initial capacity of the collection will be 0. Minimal amounts of memory will be allocated when objects are added.

This method also sets the receiver's sort selector to *theSelector* . This selector is used to compare objects and keeps the collection constantly sorted. See **setSortSelector:** for a discussion on how the selector is used.

Note that if you use FCCollection's - **init** or - **initForClass:** instead of one of FCSortedCollection's - **init...** methods, the sort selector will default to **compare:** .

See also: **- setSortSelector, - initWithSortSelector:, - sortByCompare:** (FCOrderedCollection)**, - setContentClass:** (FCCollection)**, - init** (FCCollection)**, - initCount:** (FCCollection)**, - initForClass:** (FCCollection)

initWithSortSelector:

- **initWithSortSelector:**(SEL)*theSelector;*

Initializes the receiver, a new FCSortedCollection object, but doesn't allocate any memory for its collection of objects. The initial capacity of the collection will be 0. Minimal amounts of memory will be allocated when objects are added.

This method also sets the receiver's sort selector to *theSelector* . This selector is used to compare objects and keeps the collection constantly sorted. See **setSortSelector:** for a discussion on how the selector is used.

The **initWithSortSelector:** method does not set a content class for the collection; it allows you to add objects of any type. If you wish to restrict the type of objects you can add use **initForClass:withSortSelector:** instead, or send **setContentClass:** to the receiver.

Note that if you use FCCollection's - **init** or - **initForClass:** instead of one of FCSortedCollection's - **init...** methods, the sort selector will default to **compare:** .

See also: - setSortSelector, - initForClass:withSortSelector:, - sortByCompare: (FCOrderedCollection)**, - setContentClass:** (FCCollection)**, - init** (FCCollection)**, - initCount:** (FCCollection)**, - initForClass:** (FCCollection)

isSorted

- (BOOL)**isSorted;**

Returns YES to indicate that, unlike FCCollection, this class and its subclasses keep all their objects constantly in sorted order.

See also: - isSorted (FCCollection)

setSortSelector:

- **setSortSelector:**(SEL)*theSelector;*

Sets the selector that will be used to compare objects in the collection, and resorts the collection by calling - **sortByCompare:** . The method - **addObject:** will insert objects in sorted order by utilizing *theSelector* .

Objects are compared by sending them the *theSelector* message, which must take an **id** (of the comparison object) as its sole argument. The return values for the *theSelector* method should be of type **FCCompareType** ; FC_COMPARE_EQUAL_TO, FC_COMPARE_GREATER_THAN, or FC_COMPARE_LESS_THAN, depending on whether the receiving object is equal to, greater than, or less than the argument object, respectively.

For example, the following method in the Employee class would allow employees to be sorted by age:

```
- (FCCompareType)compareAge:employee
{
   return compareInts([self age], [employee age]);
}
```

If you had a FCSortedCollection subclass of employees, you could then tell it to keep the collection sorted by age by calling:

```
[employeesByAge setSortSelector:@selector(compareAge:)]
```

If the sort selector is never set (either with - **initWithSortSelector:** , - **initForClass:withSortSelector:** , or this method), the default message sent to compare objects will be - **compare:** .

See also: **- sortSelector, - initWithSortSelector:, - initForClass:withSortSelector:, - sortByCompare:** (FCOrderedCollection)**,**

sort

- **sort;**

Sorts the collection by calling - **sortByCompare:** with the current sort selector. Since this class maintains its collection in sorted order when you delete or add objects, this method will rarely need to be called. One instance in which you will have to call this method is if you modify the sorting value of an object while it's still in the collection.

See also: **- sortByCompare:, - setSortSelector:, - initWithSortSelector:**

sortByCompare:

- **sortByCompare:**(SEL)*theSelector;*

Sets the current sort selector to be *theSelector* and sorts the objects in the collection by calling FCOrderedCollection's - **sortByCompare:** . Does nothing and returns **nil** if *theSelector* is **NULL** .

See also: **- sortByCompare:** (FCOrderedCollection)**, - setSortSelector:**

sortSelector

- (SEL)**sortSelector;**

Returns the selector that will be used to compare objects in the collection. See - **setSortSelector:** for a description of how this selector is used.

See also: **- setSortSelector:, sortByCompare:**