Sort Descriptor Programming Topics

Developer

Contents

Tables and Listings

Introduction to Sort Descriptors

Important: This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.

Sort descriptors specify how a collection of objects is sorted.

At a Glance

This programming topic contains the following article:

- Creating and Using Sort Descriptors (page 5) describes how to create and use sort descriptors.

Creating and Using Sort Descriptors

A sort descriptor describes a comparison used to sort a collection of objects. You create an instance of `NSSortDescriptor` that specifies the property key to be sorted, and whether the comparison should be in ascending, or descending order. A sort descriptor can also specify a method to use when comparing the property key values, rather than the default of `compare:`.

It is important to remember that `NSSortDescriptor` does not sort objects. It provides the description of how to sort objects. The actual sorting is done by other classes, often `NSArray` or `NSMutableArray`.

Specifying Sorts Using NSSortDescriptor

Let's assume, as an example, that we have an array (an instance of `NSArray`) containing instances of a custom class, `Employee` (that meets the requirements set out in Requirements of Collection Objects (page 7)). The `Employee` class has attributes for an employee's first and last name (instances of `NSString`), date of hire (an instance of `NSDate`), and age (an instance of `NSNumber`).

Our first task is to return an `NSArray` object sorted using the age. The example in Listing 1 illustrates how to create an `NSSortDescriptor` that can be used to sort the array contents in ascending order by the `age` key.

Listing 1 Sorting the array by the age key

```
NSSortDescriptor *ageDescriptor = [[NSSortDescriptor alloc] initWithKey:@"age"
ascending:YES];
NSArray *sortDescriptors = @[ageDescriptor];
NSArray *sortedArray = [employeesArray sortedArrayUsingDescriptors:sortDescriptors];
```

You'll note that when sorting the array it was necessary to provide an array of `NSSortDescriptor` instances. Each of the sort descriptors are applied in sequence, providing a means of sorting on multiple property keys.

If we also wanted to sort by the date of hire, we can add another descriptor to the array we provide to `sortedArrayUsingDescriptors:`. The example in Listing 2 demonstrates using multiple sort descriptors to sort on the age, and then sort employees of the same age by their date of hire.

Listing 2 Sorting the array by the age and date of hire keys

```
NSSortDescriptor *ageDescriptor = [[NSSortDescriptor alloc] initWithKey:@"age"
ascending:YES];
NSSortDescriptor *hireDateDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"hireDate" ascending:YES];
NSArray *sortDescriptors = @[ageDescriptor, hireDateDescriptor];
NSArray *sortedArray = [employeesArray sortedArrayUsingDescriptors:sortDescriptors];
```

In each of these cases, the default comparison method, `compare:`, is used. When sorting by age (where the age values are instances of `NSNumber`), the `compare:` method implemented by `NSNumber` is used; when sorting by date of hire (where the date of hire values are instances of `NSDate`), the `compare:` method implemented by `NSDate` is used.

If we want to sort the employees by name, however, since the names are strings the results should be ordered alphabetically according to the user's locale, and perhaps without case sensitivity. The default `compare:` method of `NSString` does not do this, so we need to specify a custom method to perform the comparison.

Specifying Custom Comparisons

The preceding examples all rely on the default `compare:` method to sort by age and date of hire. Names are strings, and when you sort strings to present to the user you should always use a localized comparison (see Searching, Comparing, and Sorting Strings in *String Programming Guide*). Often you also want to perform a case insensitive comparison. The example in Listing 3 shows how to specify a suitable comparison method (`localizedStandardCompare:`) to order the array by last and first name.

Listing 3 Sorting the array using a localized standard comparison

```
NSSortDescriptor *lastNameDescriptor = [[NSSortDescriptor alloc]
              initWithKey:@"lastName" ascending:YES
selector:@selector(localizedStandardCompare:)];
NSSortDescriptor * firstNameDescriptor = [[NSSortDescriptor alloc]
              initWithKey:@"firstName" ascending:YES
selector:@selector(localizedStandardCompare:)];
NSArray *sortDescriptors = @[lastNameDescriptor, firstNameDescriptor];
NSArray *sortedArray = [peopleArray sortedArrayUsingDescriptors:sortDescriptors];
```

The Foundation classes that have methods that can be used with sort descriptors are listed in Table 1.

Table 1 Common Foundation classes and comparison methods

Comparison Method	Supporting Classes
`compare:`	`NSString`, `NSMutableString`, `NSDate`, `NSCalendarDate`, `NSValue` (scalar types and unsigned char only), `NSNumber`
`caseInsensitiveCompare:`	`NSString`, `NSMutableString`
`localizedCompare:`	`NSString`, `NSMutableString`
`localizedCaseInsensitiveCompare:`	`NSString`, `NSMutableString`
`localizedStandardCompare:`	`NSString`, `NSMutableString`

You can add comparison support to their classes by implementing a compliant compare method as described in Requirements of Collection Objects.

Requirements of Collection Objects

In order for a collection to be able to sort its contents using `NSSortDescriptor`, the objects must conform to the following expectations.

- Each object in the collection must be key-value coding-compliant for the property key used to create the sort descriptor (for more about key-value coding, see *Key-Value Coding Programming Guide*).
- The object at the specified property key, relative to each object in the collection, must implement the compare selector used to create the sort descriptor. If no custom selector was specified, the objects must implement `compare:`.
- The selector used for the comparison is passed a single parameter, the object to compare against `self`, and must return the appropriate `NSComparisonResult`.

Attempting to sort a collection containing objects that fail any of these requirements will raise an exception.

Document Revision History

This table describes the changes to *Sort Descriptor Programming Topics* .

Date	Notes
2012-07-17	Updated to use modern Objective-C features.
2007-07-10	Changed string-based examples to use localized comparisons.
2003-08-08	First release of conceptual and task material covering the usage of new classes in OS X v10.3 for specifying collection sorting.