
NSFetchRequest Class Reference

Data Management



2010-05-04



Apple Inc.
© 2010 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, iPhone, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

NSFetchRequest Class Reference 5

Overview	5
Tasks	6
Entity	6
Fetch Constraints	6
Sorting	7
Prefetching	7
Managing How Results Are Returned	7
Instance Methods	8
affectedStores	8
entity	8
fetchBatchSize	8
fetchLimit	9
fetchOffset	10
includesPendingChanges	10
includesPropertyValues	11
includesSubentities	11
predicate	12
propertiesToFetch	12
relationshipKeyPathsForPrefetching	13
resultType	13
returnsDistinctResults	14
returnsObjectsAsFaults	14
setAffectedStores:	15
setEntity:	15
setFetchBatchSize:	16
setFetchLimit:	16
setFetchOffset:	16
setIncludesPendingChanges:	17
setIncludesPropertyValues:	17
setIncludesSubentities:	18
setPredicate:	18
setPropertiesToFetch:	18
setRelationshipKeyPathsForPrefetching:	19
setResultType:	19
setReturnsDistinctResults:	20
setReturnsObjectsAsFaults:	20
setSortDescriptors:	21
sortDescriptors	21
Constants	22
NSFetchRequestResultType	22

Document Revision History 23

NSFetchRequest Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.framework
Availability	Available in iOS 3.0 and later.
Declared in	NSFetchRequest.h
Companion guides	Core Data Programming Guide Predicate Programming Guide

Overview

The `NSFetchRequest` class is used to describe search criteria used to retrieve data from a persistent store.

An instance collects the criteria needed to select and—optionally—order a group of persistent objects, whether from a repository such as a file or an in-memory store such as a managed object context. A fetch request contains the following elements:

- An entity description (an instance of `NSEntityDescription`) that specifies which entity to search, and hence what type of object (if any) will be returned. This is the only mandatory element.
- A predicate (an instance of `NSPredicate`) that specifies which properties to select by and the constraints on selection, for example “last name begins with a J”. If you don’t specify a predicate, then all instances of the specified entity are selected (subject to other constraints, see `executeFetchRequest:error:` for full details).
- An array of sort descriptors (instances of `NSSortDescriptor`) that specify how the returned objects should be ordered, for example by last name then by first name.

You can also specify other aspects of a fetch request—the maximum number of objects that a request should return, and which data stores the request should access. With Mac OS X v10.5 and later you can also specify, for example, whether the fetch returns managed objects or just object IDs, and whether objects are fully populated with their properties (see [resultType](#) (page 13), [includesSubentities](#) (page 11), [includesPropertyValues](#) (page 11), and [returnsObjectsAsFaults](#) (page 14)). With Mac OS X v10.6 and later and on iOS, you can further specify, for example, what properties to fetch, the fetch offset, and whether, when the fetch is executed it matches against currently unsaved changes in the managed object context (see [resultType](#) (page 13), [propertiesToFetch](#) (page 12), [fetchOffset](#) (page 10), and [includesPendingChanges](#) (page 10)).

You use `NSFetchRequest` objects with the method `executeFetchRequest:error:`, defined by `NSManagedObjectContext`.

You often predefine fetch requests in a managed object model—`NSManagedObjectContext` provides API to retrieve a stored fetch request by name. Stored fetch requests can include placeholders for variable substitution, and so serve as templates for later completion. Fetch request templates therefore allow you to pre-define queries with variables that are substituted at runtime.

Tasks

Entity

- [entity](#) (page 8)
Returns the entity specified for the receiver.
- [setEntity:](#) (page 15)
Sets the entity of the receiver.
- [includesSubentities](#) (page 11)
Returns a Boolean value that indicates whether the receiver includes subentities in the results.
- [setIncludesSubentities:](#) (page 18)
Sets whether the receiver includes subentities.

Fetch Constraints

- [predicate](#) (page 12)
Returns the predicate of the receiver.
- [setPredicate:](#) (page 18)
Sets the predicate of the receiver.
- [fetchLimit](#) (page 9)
Returns the fetch limit of the receiver.
- [setFetchLimit:](#) (page 16)
Sets the fetch limit of the receiver.
- [fetchOffset](#) (page 10)
Returns the fetch offset of the receiver.
- [setFetchOffset:](#) (page 16)
Sets the fetch offset of the receiver.
- [fetchBatchSize](#) (page 8)
Returns the batch size of the receiver.
- [setFetchBatchSize:](#) (page 16)
Sets the fetch offset of the receiver.
- [affectedStores](#) (page 8)
Returns an array containing the persistent stores specified for the receiver.
- [setAffectedStores:](#) (page 15)
Sets the array of persistent stores that will be searched by the receiver.

Sorting

- [sortDescriptors](#) (page 21)
Returns the sort descriptors of the receiver.
- [setSortDescriptors:](#) (page 21)
Sets the array of sort descriptors of the receiver.

Prefetching

- [relationshipKeyPathsForPrefetching](#) (page 13)
Returns the array of relationship keypaths to prefetch along with the entity for the request.
- [setRelationshipKeyPathsForPrefetching:](#) (page 19)
Sets an array of relationship keypaths to prefetch along with the entity for the request.

Managing How Results Are Returned

- [resultType](#) (page 13)
Returns the result type of the receiver.
- [setResultType:](#) (page 19)
Sets the result type of the receiver.
- [includesPendingChanges](#) (page 10)
Returns a Boolean value that indicates whether, when the fetch is executed it matches against currently unsaved changes in the managed object context.
- [setIncludesPendingChanges:](#) (page 17)
Sets if, when the fetch is executed, it matches against currently unsaved changes in the managed object context.
- [propertiesToFetch](#) (page 12)
Returns an array of `NSPropertyDescription` objects that specify which properties should be returned by the fetch.
- [setPropertyToFetch:](#) (page 18)
Specifies which properties should be returned by the fetch.
- [returnsDistinctResults](#) (page 14)
Returns a Boolean value that indicates whether the fetch request returns only distinct values for the fields specified by `propertiesToFetch`.
- [setReturnsDistinctResults:](#) (page 20)
Sets whether the request should return only distinct values for the fields specified by `propertiesToFetch`.
- [includesPropertyValues](#) (page 11)
Returns a Boolean value that indicates whether, when the fetch is executed, property data is obtained from the persistent store.
- [setIncludesPropertyValues:](#) (page 17)
Sets if, when the fetch is executed, property data is obtained from the persistent store.
- [returnsObjectsAsFaults](#) (page 14)
Returns a Boolean value that indicates whether the objects resulting from a fetch using the receiver are faults.

- [setReturnsObjectsAsFaults:](#) (page 20)
Sets whether the objects resulting from a fetch request are faults.

Instance Methods

affectedStores

Returns an array containing the persistent stores specified for the receiver.

- (NSArray *)affectedStores

Return Value

An array containing the persistent stores specified for the receiver.

Availability

Available in iOS 3.0 and later.

See Also

- [setAffectedStores:](#) (page 15)

Declared In

NSFetchRequest.h

entity

Returns the entity specified for the receiver.

- (NSEntityDescription *)entity

Return Value

The entity specified for the receiver.

Availability

Available in iOS 3.0 and later.

See Also

- [setEntity:](#) (page 15)

Declared In

NSFetchRequest.h

fetchBatchSize

Returns the batch size of the receiver.

- (NSUInteger)fetchBatchSize

Return Value

The batch size of the receiver.

Discussion

The default value is 0. A batch size of 0 is treated as infinite, which disables the batch faulting behavior.

If you set a non-zero batch size, the collection of objects returned when the fetch is executed is broken into batches. When the fetch is executed, the entire request is evaluated and the identities of all matching objects recorded, but no more than *batchSize* objects' data will be fetched from the persistent store at a time. The array returned from executing the request will be a proxy object that transparently faults batches on demand. (In database terms, this is an in-memory cursor.)

You can use this feature to restrict the working set of data in your application. In combination with [fetchLimit](#) (page 9), you can create a subrange of an arbitrary result set.

Special Considerations

For purposes of thread safety, you should consider the array proxy returned when the fetch is executed as being owned by the managed object context the request is executed against, and treat it as if it were a managed object registered with that context.

Availability

Available in iOS 3.0 and later.

See Also

- [setFetchBatchSize](#): (page 16)
- [fetchLimit](#) (page 9)

Declared In

NSFetchRequest.h

fetchLimit

Returns the fetch limit of the receiver.

```
- (NSUInteger)fetchLimit
```

Return Value

The fetch limit of the receiver.

Discussion

The fetch limit specifies the maximum number of objects that a request should return when executed.

Special Considerations

If you set a fetch limit, the framework makes a best effort, but does not guarantee, to improve efficiency. For every object store except the SQL store, a fetch request executed with a fetch limit in effect simply performs an unlimited fetch and throws away the unasked for rows.

Availability

Available in iOS 3.0 and later.

See Also

- [setFetchLimit](#): (page 16)
- [fetchOffset](#) (page 10)

Declared In

NSFetchRequest.h

fetchOffset

Returns the fetch offset of the receiver.

- (NSUInteger)fetchOffset

Return Value

The fetch offset of the receiver.

Discussion

The default value is 0.

This setting allows you to specify an offset at which rows will begin being returned. Effectively, the request will skip over the specified number of matching entries. For example, given a fetch which would normally return a, b, c, d, specifying an offset of 1 will return b, c, d, and an offset of 4 will return an empty array. Offsets are ignored in nested requests such as subqueries.

This can be used to restrict the working set of data. In combination with `-fetchLimit`, you can create a subrange of an arbitrary result set.

Availability

Available in iOS 3.0 and later.

See Also

- [setFetchOffset:](#) (page 16)
- [fetchLimit](#) (page 9)

Declared In

NSFetchRequest.h

includesPendingChanges

Returns a Boolean value that indicates whether, when the fetch is executed it matches against currently unsaved changes in the managed object context.

- (BOOL)includesPendingChanges

Return Value

YES if, when the fetch is executed it will match against currently unsaved changes in the managed object context, otherwise NO.

Discussion

The default value is YES.

If the value is NO, the fetch request skips checking unsaved changes and only returns objects that matched the predicate in the persistent store.

Availability

Available in iOS 3.0 and later.

See Also

- [setIncludesPendingChanges:](#) (page 17)

Declared In

NSFetchRequest.h

includesPropertyValues

Returns a Boolean value that indicates whether, when the fetch is executed, property data is obtained from the persistent store.

- (BOOL)includesPropertyValues

Return Value

YES if, when the fetch is executed, property data is obtained from the persistent store, otherwise NO.

Discussion

The default value is YES.

You can set `includesPropertyValues` to NO to reduce memory overhead by avoiding creation of objects to represent the property values. You should typically only do so, however, if you are sure that either you will not need the actual property data or you already have the information in the row cache, otherwise you will incur multiple trips to the database.

During a normal fetch (`includesPropertyValues` is YES), Core Data fetches the object ID *and* property data for the matching records, fills the row cache with the information, and returns managed object as faults (see [returnsObjectsAsFaults](#) (page 14)). These faults are managed objects, but all of their property data still resides in the row cache until the fault is fired. When the fault is fired, Core Data retrieves the data from the row cache—there is no need to go back to the database.

If `includesPropertyValues` is NO, then Core Data fetches *only* the object ID information for the matching records—it does not populate the row cache. Core Data still returns managed objects since it only needs managed object IDs to create faults. However, if you subsequently fire the fault, Core Data looks in the (empty) row cache, doesn't find any data, and then goes back to the store a second time for the data.

Availability

Available in iOS 3.0 and later.

See Also

- [setIncludesPropertyValues:](#) (page 17)

Declared In

NSFetchRequest.h

includesSubentities

Returns a Boolean value that indicates whether the receiver includes subentities in the results.

- (BOOL)includesSubentities

Return Value

YES if the request will include all subentities of the entity for the request, otherwise NO.

Discussion

The default is YES.

Availability

Available in iOS 3.0 and later.

See Also

- [setIncludesSubentities:](#) (page 18)

Declared In

NSFetchRequest.h

predicate

Returns the predicate of the receiver.

- (NSPredicate *)predicate

Return Value

The predicate of the receiver.

Discussion

The predicate is used to constrain the selection of objects the receiver is to fetch. For more about predicates, see *Predicate Programming Guide*.

If the predicate is empty—for example, if it is an AND predicate whose array of elements contains no predicates—the receiver has its predicate set to `nil`. For more about predicates, see *Predicate Programming Guide*.

Availability

Available in iOS 3.0 and later.

See Also- [setPredicate:](#) (page 18)**Declared In**

NSFetchRequest.h

propertiesToFetchReturns an array of `NSPropertyDescription` objects that specify which properties should be returned by the fetch.

- (NSArray *)propertiesToFetch

Return ValueAn array of `NSPropertyDescription` objects that specify which properties should be returned by the fetch.**Discussion**For a full discussion, see [setPropertyToFetch:](#) (page 18).**Availability**

Available in iOS 3.0 and later.

See Also- [setPropertyToFetch:](#) (page 18)- [resultType](#) (page 13)- [returnsDistinctResults](#) (page 14)**Declared In**

NSFetchRequest.h

relationshipKeyPathsForPrefetching

Returns the array of relationship keypaths to prefetch along with the entity for the request.

- (NSArray *)relationshipKeyPathsForPrefetching

Return Value

The array of relationship keypaths to prefetch along with the entity for the request.

Discussion

The default value is an empty array (no prefetching).

Prefetching allows Core Data to obtain related objects in a single fetch (per entity), rather than incurring subsequent access to the store for each individual record as their faults are tripped. For example, given an Employee entity with a relationship to a Department entity, if you fetch all the employees then for each print out their name and the name of the department to which they belong, it may be that a fault has to be fired for each individual Department object (for more details, see “Core Data Performance” in *Core Data Programming Guide*). This can represent a significant overhead. You could avoid this by prefetching the department relationship in the Employee fetch, as illustrated in the following example:

```
NSManagedObjectContext *context = ...;
NSEntityDescription *employeeEntity = [NSEntityDescription
    entityForName:@"Employee" inManagedObjectContext:context];
NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity:employeeEntity];
[request setRelationshipKeyPathsForPrefetching:
    [NSArray arrayWithObject:@"department"]];
```

Availability

Available in iOS 3.0 and later.

See Also

- [setRelationshipKeyPathsForPrefetching:](#) (page 19)

Declared In

NSFetchRequest.h

resultType

Returns the result type of the receiver.

- (NSFetchRequestResultType)resultType

Return Value

The result type of the receiver.

Discussion

The default value is `NSManagedObjectResultType`.

You use [setResultType:](#) (page 19) to set the instance type of objects returned from executing the request—for possible values, see “Fetch request result types” (page 22). If you set the value to `NSManagedObjectIDResultType`, this will demote any sort orderings to “best efforts” hints if you do not include the property values in the request.

Availability

Available in iOS 3.0 and later.

See Also

- [resultSetType:](#) (page 19)

Declared In

NSFetchRequest.h

returnsDistinctResults

Returns a Boolean value that indicates whether the fetch request returns only distinct values for the fields specified by `propertiesToFetch`.

- (BOOL)returnsDistinctResults

Return Value

YES if, when the fetch is executed, it returns only distinct values for the fields specified by `propertiesToFetch`, otherwise NO.

Discussion

The default value is NO.

Special Considerations

This value is only used if a value has been set for [propertiesToFetch](#) (page 12).

Availability

Available in iOS 3.0 and later.

See Also

- [setReturnsDistinctResults:](#) (page 20)
- [propertiesToFetch](#) (page 12)

Declared In

NSFetchRequest.h

returnsObjectsAsFaults

Returns a Boolean value that indicates whether the objects resulting from a fetch using the receiver are faults.

- (BOOL)returnsObjectsAsFaults

Return Value

YES if the objects resulting from a fetch using the receiver are faults, otherwise NO.

Discussion

The default value is YES. This setting is not used if the result type (see [resultSetType](#) (page 13)) is `NSManagedObjectIDResultType`, as object IDs do not have property values. You can set `returnsObjectsAsFaults` to NO to gain a performance benefit if you know you will need to access the property values from the returned objects.

By default, when you execute a fetch `returnsObjectsAsFaults` is YES; Core Data fetches the object data for the matching records, fills the row cache with the information, and returns managed object as faults. These faults are managed objects, but all of their property data resides in the row cache until the fault is fired. When the fault is fired, Core Data retrieves the data from the row cache. Although the overhead for this operation is small for large datasets it may become non-trivial. If you *need* to access the property values

from the returned objects (for example, if you iterate over all the objects to calculate the average value of a particular attribute), then it is more efficient to set `returnsObjectsAsFaults` to `NO` to avoid the additional overhead.

Availability

Available in iOS 3.0 and later.

See Also

- [setReturnsObjectsAsFaults:](#) (page 20)

Declared In

NSFetchRequest.h

setAffectedStores:

Sets the array of persistent stores that will be searched by the receiver.

```
- (void)setAffectedStores:(NSArray *)stores
```

Parameters

stores

An array containing identifiers for the stores to be searched when the receiver is executed.

Availability

Available in iOS 3.0 and later.

See Also

- [affectedStores](#) (page 8)

Declared In

NSFetchRequest.h

setEntity:

Sets the entity of the receiver.

```
- (void)setEntity:(NSEntityDescription *)entity
```

Parameters

entity

The entity of the receiver.

Availability

Available in iOS 3.0 and later.

See Also

- [entity](#) (page 8)

Declared In

NSFetchRequest.h

setFetchBatchSize:

Sets the fetch offset of the receiver.

- (void)setFetchBatchSize:(NSUInteger)*batchSize*

Parameters

batchSize

The batch size of the receiver.

A batch size of 0 is treated as infinite, which disables the batch faulting behavior.

Discussion

For a full discussion, see [fetchBatchSize](#) (page 8).

Availability

Available in iOS 3.0 and later.

See Also

- [fetchBatchSize](#) (page 8)

- [fetchLimit](#) (page 9)

Declared In

NSFetchRequest.h

setFetchLimit:

Sets the fetch limit of the receiver.

- (void)setFetchLimit:(NSUInteger)*limit*

Parameters

limit

The fetch limit of the receiver. 0 specifies no fetch limit.

Discussion

For a full discussion, see [fetchLimit](#) (page 9).

Availability

Available in iOS 3.0 and later.

See Also

- [fetchLimit](#) (page 9)

- [fetchOffset](#) (page 10)

Declared In

NSFetchRequest.h

setFetchOffset:

Sets the fetch offset of the receiver.

- (void)setFetchOffset:(NSUInteger)*offset*

Parameters*limit*

The fetch offset of the receiver.

Discussion

For a full discussion, see [fetchOffset](#) (page 10).

Availability

Available in iOS 3.0 and later.

See Also

- [fetchOffset](#) (page 10)

- [fetchLimit](#) (page 9)

Declared In

NSFetchRequest.h

setIncludesPendingChanges:

Sets if, when the fetch is executed, it matches against currently unsaved changes in the managed object context.

- (void)setIncludesPendingChanges:(BOOL)yesNo

Parameters*yesNo*

If YES, when the fetch is executed it will match against currently unsaved changes in the managed object context.

Discussion

For a full discussion, see [includesPendingChanges](#) (page 10).

Availability

Available in iOS 3.0 and later.

See Also

- [includesPendingChanges](#) (page 10)

Declared In

NSFetchRequest.h

setIncludesPropertyValues:

Sets if, when the fetch is executed, property data is obtained from the persistent store.

- (void)setIncludesPropertyValues:(BOOL)yesNo

Parameters*yesNo*

If NO, the request will not obtain property information, but only information to identify each object (used to create managed object IDs).

Discussion

For a full discussion, see [includesPropertyValues](#) (page 11).

Availability

Available in iOS 3.0 and later.

See Also

- [includesPropertyValues](#) (page 11)

Declared In

NSFetchRequest.h

setIncludesSubentities:

Sets whether the receiver includes subentities.

- (void)setIncludesSubentities:(BOOL)*yesNo*

Parameters

yesNo

If NO, the receiver will fetch objects of exactly the entity type of the request; if YES, the receiver will include all subentities of the entity for the request (if any).

Availability

Available in iOS 3.0 and later.

See Also

- [includesSubentities](#) (page 11)

Declared In

NSFetchRequest.h

setPredicate:

Sets the predicate of the receiver.

- (void)setPredicate:(NSPredicate *)*predicate*

Parameters

predicate

The predicate for the receiver.

Availability

Available in iOS 3.0 and later.

See Also

- [predicate](#) (page 12)

Declared In

NSFetchRequest.h

setPropertyToFetch:

Specifies which properties should be returned by the fetch.

- (void)setPropertiesToFetch:(NSArray *)*values*

Parameters

values

An array of `NSPropertyDescription` objects that specify which properties should be returned by the fetch.

Discussion

The property descriptions may represent attributes, to-one relationships, or expressions. The name of an attribute or relationship description must match the name of a description on the fetch request's entity.

Availability

Available in iOS 3.0 and later.

See Also

- [propertiesToFetch](#) (page 12)
- [resultType](#) (page 13)
- [returnsDistinctResults](#) (page 14)

Declared In

NSFetchRequest.h

setRelationshipKeyPathsForPrefetching:

Sets an array of relationship keypaths to prefetch along with the entity for the request.

- (void)setRelationshipKeyPathsForPrefetching:(NSArray *)*keys*

Parameters

keys

An array of relationship key-path strings in `NSKeyValueCoding` notation (as you would normally use with `valueForKeyPath:`).

Discussion

For a full discussion, see [relationshipKeyPathsForPrefetching](#) (page 13).

Availability

Available in iOS 3.0 and later.

See Also

- [relationshipKeyPathsForPrefetching](#) (page 13)

Declared In

NSFetchRequest.h

setResultType:

Sets the result type of the receiver.

- (void)setResultType:(NSFetchRequestResultType)*type*

Parameters*type*

The result type of the receiver.

DiscussionFor further discussion, see [resultType](#) (page 13).**Availability**

Available in iOS 3.0 and later.

See Also- [resultType](#) (page 13)**Declared In**

NSFetchRequest.h

setReturnsDistinctResults:Sets whether the request should return only distinct values for the fields specified by `propertiesToFetch`.- (void)setReturnsDistinctResults:(BOOL) *values***Parameters***values*If YES, the request returns only distinct values for the fields specified by `propertiesToFetch`.**Discussion**For a full discussion, see [returnsDistinctResults](#) (page 14).**Special Considerations**This value is only used if a value has been set for `propertiesToFetch`.**Availability**

Available in iOS 3.0 and later.

See Also- [returnsDistinctResults](#) (page 14)- [propertiesToFetch](#) (page 12)**Declared In**

NSFetchRequest.h

setReturnsObjectsAsFaults:

Sets whether the objects resulting from a fetch request are faults.

- (void)setReturnsObjectsAsFaults:(BOOL) *yesNo*

Parameters*yesNo*

If `NO`, the objects returned from the fetch are pre-populated with their property values (making them fully-faulted objects, which will immediately return `NO` if sent the `isFault` message). If `YES`, the objects returned from the fetch are not pre-populated (and will receive a `didFireFault` message when the properties are accessed the first time).

Discussion

For a full discussion, see [returnsObjectsAsFaults](#) (page 14).

Availability

Available in iOS 3.0 and later.

See Also

- [returnsObjectsAsFaults](#) (page 14)

Declared In

NSFetchRequest.h

setSortDescriptors:

Sets the array of sort descriptors of the receiver.

```
- (void)setSortDescriptors:(NSArray *)sortDescriptors
```

Parameters*sortDescriptors*

The array of sort descriptors of the receiver. `nil` specifies no sort descriptors.

Availability

Available in iOS 3.0 and later.

See Also

- [sortDescriptors](#) (page 21)

Declared In

NSFetchRequest.h

sortDescriptors

Returns the sort descriptors of the receiver.

```
- (NSArray *)sortDescriptors
```

Return Value

The sort descriptors of the receiver.

Discussion

The sort descriptors specify how the objects returned when the fetch request is issued should be ordered—for example by last name then by first name. The sort descriptors are applied in the order in which they appear in the *sortDescriptors* array (serially in lowest-array-index-first order).

Availability

Available in iOS 3.0 and later.

See Also

- [setSortDescriptors:](#) (page 21)

Declared In

NSFetchRequest.h

Constants

NSFetchRequestResultType

These constants specify the possible result types a fetch request can return.

```
enum {
    NSManagedObjectResultType      = 0x00,
    NSManagedObjectIDResultType    = 0x01,
    NSDictionaryResultType          = 0x02
};
typedef NSUInteger NSFetchRequestResultType;
```

Constants

`NSManagedObjectResultType`
Specifies that the request returns managed objects.
 Available in iOS 3.0 and later.
 Declared in `NSFetchRequest.h`.

`NSManagedObjectIDResultType`
Specifies that the request returns managed object IDs.
 Available in iOS 3.0 and later.
 Declared in `NSFetchRequest.h`.

`NSDictionaryResultType`
Specifies that the request returns dictionaries.
 Available in iOS 3.0 and later.
 Declared in `NSFetchRequest.h`.

Discussion

These constants are used by [resultType](#) (page 13).

Document Revision History

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

Date	Notes
2010-05-04	Formatting change.
2010-02-24	Corrected error in abstract of <code>setReturnsDistinctResults:</code> .
2009-08-06	Corrected a minor typographical error.
2009-06-15	Corrected description of <code>setPropertyTypesToFetch</code> .
2009-02-19	Updated for iOS 3.0.
2008-10-03	Updated for Mac OS X v10.6.
2008-02-08	Corrected the discussion of the <code>resultType</code> method.
2006-07-09	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a separate document.

REVISION HISTORY

Document Revision History