
NSManagedObjectContext Class Reference

[Cocoa > Data Management](#)



2008-10-15



Apple Inc.
© 2008 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, Cocoa, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

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

NSManagedObjectContext Class Reference 5

Overview	5
Life-cycle Management	5
Persistent Store Coordinator	6
Subclassing Notes	6
Tasks	6
Registering and Fetching Objects	6
Managed Object Management	6
Merging Changes from Another Context	7
Undo Management	7
Locking	8
Delete Propagation	8
Retaining Registered Objects	8
Managing the Persistent Store Coordinator	8
Managing the Staleness Interval	8
Managing the Merge Policy	9
Supporting NSKeyValueObserving Protocol	9
Supporting NSEditor and NSEditorRegistration Protocols	9
Instance Methods	9
assignObject:toPersistentStore:	9
commitEditing	10
commitEditingWithDelegate:didCommitSelector:contextInfo:	11
countForFetchRequest:error:	12
deletedObjects	12
deleteObject:	13
detectConflictsForObject:	14
discardEditing	14
executeFetchRequest:error:	14
hasChanges	15
insertedObjects	16
insertObject:	16
lock	17
mergeChangesFromContextDidSaveNotification:	17
mergePolicy	18
objectDidBeginEditing:	18
objectDidEndEditing:	19
objectRegisteredForID:	19
objectWithID:	20
observeValueForKeyPath:ofObject:change:context:	20
obtainPermanentIDsForObjects:error:	21
persistentStoreCoordinator	22

- processPendingChanges 22
- propagatesDeletesAtEndOfEvent 22
- redo 23
- refreshObject:mergeChanges: 23
- registeredObjects 24
- reset 25
- retainsRegisteredObjects 25
- rollback 25
- save: 26
- setMergePolicy: 26
- setPersistentStoreCoordinator: 27
- setPropagatesDeletesAtEndOfEvent: 27
- setRetainsRegisteredObjects: 28
- setStalenessInterval: 28
- setUndoManager: 29
- stalenessInterval 29
- tryLock 30
- undo 30
- undoManager 30
- unlock 31
- updatedObjects 31
- Constants 32
 - NSManagedObjectContext Change Notification User Info Keys 32
 - Merge Policies 33
- Notifications 35
 - NSManagedObjectContextObjectsDidChangeNotification 35
 - NSManagedObjectContextDidSaveNotification 35

Document Revision History 37

Index 39

NSManagedObjectContext Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.framework
Availability	Available in Mac OS X v10.4 and later.
Declared in	NSManagedObjectContext.h
Companion guides	Core Data Programming Guide NSPersistentDocument Core Data Tutorial Core Data Utility Tutorial Predicate Programming Guide
Related sample code	Core Data HTML Store CoreRecipes Departments and Employees QTMetadataEditor

Overview

An instance of `NSManagedObjectContext` represents a single “object space” or scratch pad in an application. Its primary responsibility is to manage a collection of managed objects. These objects form a group of related model objects that represent an internally consistent view of one or more persistent stores. A single managed object instance exists in one and only one context, but multiple copies of an object can exist in different contexts. Thus object uniquing is scoped to a particular context.

Life-cycle Management

The context is a powerful object with a central role in the life-cycle of managed objects, with responsibilities from life-cycle management (including faulting) to validation, inverse relationship handling, and undo/redo. Through a context you can retrieve or “fetch” objects from a persistent store, make changes to those objects, and then either discard the changes or—again through the context—commit them back to the persistent store. The context is responsible for watching for changes in its objects and maintains an undo manager so you can have finer-grained control over undo and redo. You can insert new objects and delete ones you have fetched, and commit these modifications to the persistent store.

If you are using Cocoa Bindings, the context can also serve as a controller. It implements the `NSEditor` and `NSEditorRegistration` informal protocols, although there should typically be little reason for you to directly invoke any method other than `commitEditing` (page 10) or `commitEditingWithDelegate:didCommitSelector:contextInfo:` (page 11), and then only rarely.

Persistent Store Coordinator

A context always has a “parent” persistent store coordinator which provides the model and dispatches requests to the various persistent stores containing the data. Without a coordinator, a context is not fully functional. The context’s coordinator provides the managed object model and handles persistency. All objects fetched from an external store are registered in a context together with a global identifier (an instance of `NSManagedObjectID`) that’s used to uniquely identify each object to the external store.

Subclassing Notes

You are strongly discouraged from subclassing `NSManagedObjectContext`. The change tracking and undo management mechanisms are highly optimized and hence intricate and delicate. Interposing your own additional logic that might impact `processPendingChanges` can have unforeseen consequences. In situations such as store migration, Core Data will create instances of `NSManagedObjectContext` for its own use. Under these circumstances, you cannot rely on any features of your custom subclass. Any `NSManagedObject` subclass must always be fully compatible with `NSManagedObjectContext` (as opposed to any subclass of `NSManagedObjectContext`).

Tasks

Registering and Fetching Objects

- `objectRegisteredForID:` (page 19)
Returns the object for a specified ID, if the object is registered with the receiver.
- `objectWithID:` (page 20)
Returns the object for a specified ID.
- `executeFetchRequest:error:` (page 14)
Returns an array of objects that meet the criteria specified by a given fetch request.
- `countForFetchRequest:error:` (page 12)
Returns the number of objects a given fetch request would have returned if it had been passed to `executeFetchRequest:error:`.
- `registeredObjects` (page 24)
Returns the set of objects registered with the receiver.

Managed Object Management

- `insertObject:` (page 16)
Registers an object to be inserted in the receiver’s persistent store the next time changes are saved.

- [deleteObject:](#) (page 13)
Specifies an object that should be removed from its persistent store when changes are committed.
- [assignObject:toPersistentStore:](#) (page 9)
Specifies the store in which a newly-inserted object will be saved.
- [obtainPermanentIDsForObjects:error:](#) (page 21)
Converts to permanent IDs the object IDs of the objects in a given array.
- [detectConflictsForObject:](#) (page 14)
Marks an object for conflict detection.
- [refreshObject:mergeChanges:](#) (page 23)
Updates the persistent properties of a managed object to use the latest values from the persistent store.
- [processPendingChanges](#) (page 22)
Forces the receiver to process changes to the object graph.
- [insertedObjects](#) (page 16)
Returns the set of objects that have been inserted into the receiver but not yet saved in a persistent store.
- [updatedObjects](#) (page 31)
Returns the set of objects registered with the receiver that have uncommitted changes.
- [deletedObjects](#) (page 12)
Returns the set of objects that will be removed from their persistent store during the next save operation.

Merging Changes from Another Context

- [mergeChangesFromContextDidSaveNotification:](#) (page 17)
Merges the changes specified in a given notification.

Undo Management

- [undoManager](#) (page 30)
Returns the undo manager of the receiver.
- [setUndoManager:](#) (page 29)
Sets the undo manager of the receiver.
- [undo](#) (page 30)
Sends an undo message to the receiver's undo manager, asking it to reverse the latest uncommitted changes applied to objects in the object graph.
- [redo](#) (page 23)
Sends a redo message to the receiver's undo manager, asking it to reverse the latest undo operation applied to objects in the object graph.
- [reset](#) (page 25)
Returns the receiver to its base state.
- [rollback](#) (page 25)
Removes everything from the undo stack, discards all insertions and deletions, and restores updated objects to their last committed values.

- [save:](#) (page 26)
Attempts to commit unsaved changes to registered objects to their persistent store.
- [hasChanges](#) (page 15)
Returns a Boolean value that indicates whether the receiver has uncommitted changes.

Locking

- [lock](#) (page 17)
Attempts to acquire a lock on the receiver.
- [unlock](#) (page 31)
Relinquishes a previously acquired lock.
- [tryLock](#) (page 30)
Attempts to acquire a lock.

Delete Propagation

- [propagatesDeletesAtEndOfEvent](#) (page 22)
Returns a Boolean that indicates whether the receiver propagates deletes at the end of the event in which a change was made.
- [setPropagatesDeletesAtEndOfEvent:](#) (page 27)
Sets whether the context propagates deletes to related objects at the end of the event.

Retaining Registered Objects

- [retainsRegisteredObjects](#) (page 25)
Returns a Boolean that indicates whether the receiver sends a `retain` message to objects upon registration.
- [setRetainsRegisteredObjects:](#) (page 28)
Sets whether or not the receiver retains all registered objects, or only objects necessary for a pending save (those that are inserted, updated, deleted, or locked).

Managing the Persistent Store Coordinator

- [persistentStoreCoordinator](#) (page 22)
Returns the persistent store coordinator of the receiver.
- [setPersistentStoreCoordinator:](#) (page 27)
Sets the persistent store coordinator of the receiver.

Managing the Staleness Interval

- [stalenessInterval](#) (page 29)
Returns the staleness interval of the receiver.

- [setStalenessInterval:](#) (page 28)
Sets the staleness interval of the receiver.

Managing the Merge Policy

- [mergePolicy](#) (page 18)
Returns the merge policy of the receiver.
- [setMergePolicy:](#) (page 26)
Sets the merge policy of the receiver.

Supporting NSKeyValueObserving Protocol

- [observeValueForKeyPath:ofObject:change:context:](#) (page 20)
This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

Supporting NSEditor and NSEditorRegistration Protocols

- [commitEditing](#) (page 10)
Returns a Boolean that indicates whether the receiver was able to commit any pending edits in known editors.
- [commitEditingWithDelegate:didCommitSelector:contextInfo:](#) (page 11)
Attempts to commit any pending changes in known editors of the receiver.
- [discardEditing](#) (page 14)
Causes the receiver to discard any changes in known editors, restoring the previous values
- [objectDidBeginEditing:](#) (page 18)
Provides support for the `NSEditorRegistration` informal protocol.
- [objectDidEndEditing:](#) (page 19)
Provides support for the `NSEditorRegistration` informal protocol.

Instance Methods

assignObject:toPersistentStore:

Specifies the store in which a newly-inserted object will be saved.

```
(void)assignObject:(id)object toPersistentStore:(NSPersistentStore *)store
```

Parameters

object

A managed object.

store

A persistent store.

Discussion

You can obtain a store from the persistent store coordinator, using for example `persistentStoreForURL:`.

Special Considerations

It is only necessary to use this method if the receiver's persistent store coordinator manages multiple writable stores that have *object's* entity in their configuration. Maintaining configurations in the managed object model can eliminate the need for invoking this method directly in many situations. If the receiver's persistent store coordinator manages only a single writable store, or if only one store has *object's* entity in its model, *object* will automatically be assigned to that store.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [insertObject:](#) (page 16)
- [persistentStoreCoordinator](#) (page 22)

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

commitEditing

Returns a Boolean that indicates whether the receiver was able to commit any pending edits in known editors.

- (BOOL)commitEditing

Return Value

YES if the changes were successfully applied, otherwise NO.

Discussion

This method attempts to commit pending changes in known *external editors*—it does *not* commit unsaved changes to registered objects to their persistent store (contrast [save:](#) (page 26)). Known editors are either instances of a subclass of `NSController` or (more rarely) user interface controls that may contain pending edits—such as text fields—that registered with the context using `objectDidBeginEditing:` and have not yet unregistered using a subsequent invocation of `objectDidEndEditing:`. Note that controllers only register with the context as being an editor if their `content` binding is *not* bound—if they have content of any kind, then they do not register.

The receiver iterates through the array of its known editors and invokes `commitEditing` on each until either it reaches the end of the array or an editor returns NO. If an editor returns NO, then the context also returns NO; otherwise the context returns YES.

You may find this method useful in some situations (typically if you are using Cocoa Bindings) when you want to ensure that pending changes are applied before a change in user interface state. For example, you may need to ensure that changes pending in a text field are applied before a window is closed. See also [commitEditingWithDelegate:didCommitSelector:contextInfo:](#) (page 11) which performs a similar function but which allows you to specify a delegate that will handle any errors—the delegate is informed which editor failed to commit, which may be useful if you want to display an alert panel on the editor's window.

See Also

- [commitEditingStyleWithDelegate:didCommitSelector:contextInfo:](#) (page 11)
- [discardEditingStyle](#) (page 14)
- [objectDidBeginEditing:](#) (page 18)
- [objectDidEndEditing:](#) (page 19)

commitEditingStyleWithDelegate:didCommitSelector:contextInfo:

Attempts to commit any pending changes in known editors of the receiver.

```
-(void)commitEditingStyleWithDelegate:(id)delegate
      didCommitSelector:(SEL)didCommitSelector contextInfo:(void *)contextInfo
```

Parameters

delegate

An object that can serve as the receiver's delegate. It should implement the method specified by *didCommitSelector*.

didCommitSelector

A selector that is invoked on delegate. The method specified by the selector must have the same signature as the following method:

```
-(void)editor:(id)editor didCommit:(BOOL)didCommit contextInfo:(void *)contextInfo
```

contextInfo

Contextual information that is sent as the *contextInfo* argument to delegate when *didCommitSelector* is invoked.

Discussion

Provides support for the `NSEditor` informal protocol. This method attempts to commit pending changes in known *external editors*—it does *not* commit unsaved changes to registered objects to their persistent store (contrast [save:](#) (page 26)). Known editors are either instances of a subclass of `NSController` or (more rarely) user interface controls that may contain pending edits—such as text fields—that registered with the context using `objectDidBeginEditing:` and have not yet unregistered using a subsequent invocation of `objectDidEndEditing:`. Note that controllers only register with the context as being an editor if their content binding is *not* bound—if they have content of any kind, then they do not register.

The receiver iterates through the array of its known editors and invokes `commitEditingStyle` on each. The receiver then sends the message specified by the *didCommitSelector* selector to the specified delegate.

The `didCommit` argument is the value returned by the editor specified by `editor` from the `commitEditingStyle` message. The `contextInfo` argument is the same value specified as the *contextInfo* parameter—you may use this value however you wish.

If an error occurs while attempting to commit, for example if key-value coding validation fails, your implementation of this method should typically send the view in which editing is being performed a `presentError:modalForWindow:delegate:didRecoverSelector:contextInfo:` message, specifying the view's containing window.

You may find this method useful in some situations (typically if you are using Cocoa Bindings) when you want to ensure that pending changes are applied before a change in user interface state. For example, you may need to ensure that changes pending in a text field are applied before a window is closed. See also [commitEditing](#) (page 10) which performs a similar function but which allows you to handle any errors directly, although it provides no information beyond simple success/failure.

See Also

- [commitEditing](#) (page 10)
- [discardEditing](#) (page 14)
- [objectDidBeginEditing:](#) (page 18)
- [objectDidEndEditing:](#) (page 19)

countForFetchRequest:error:

Returns the number of objects a given fetch request would have returned if it had been passed to `executeFetchRequest:error:`.

```
- (NSUInteger)countForFetchRequest:(NSFetchRequest *)request error:(NSError **)error
```

Parameters

request

A fetch request that specifies the search criteria for the fetch.

error

If there is a problem executing the fetch, upon return contains an instance of `NSError` that describes the problem.

Return Value

The number of objects a given fetch request would have returned if it had been passed to [executeFetchRequest:error:](#) (page 14). If an error occurs during the processing of the request, returns `NSNotFound`.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSManagedObjectContext.h`

deletedObjects

Returns the set of objects that will be removed from their persistent store during the next save operation.

```
- (NSSet *)deletedObjects
```

Return Value

The set of objects that will be removed from their persistent store during the next save operation.

Discussion

Note that the returned set does not necessarily include all the objects that have been deleted (using [deleteObject:](#) (page 13))—if an object has been inserted and deleted without an intervening save operation, it is not included in the set.

A managed object context does not post key-value observing notifications when the return value of `deletedObjects` changes—it does, however, post a [NSManagedObjectContextObjectsDidChangeNotification](#) (page 35) notification when a change is made, and a [NSManagedObjectContextDidSaveNotification](#) (page 35) notification when changes are committed (although again note that the set of deleted objects given for a [NSManagedObjectContextDidSaveNotification](#) (page 35) does not include objects that were inserted and deleted without an intervening save operation—that is, they had not been saved to a persistent store).

Availability

Available in Mac OS X v10.4 and later.

See Also

- [deleteObject:](#) (page 13)
- [insertedObjects](#) (page 16)
- [registeredObjects](#) (page 24)
- [updatedObjects](#) (page 31)
- `isDeleted` (`NSManagedObject`)

Declared In

`NSManagedObjectContext.h`

deleteObject:

Specifies an object that should be removed from its persistent store when changes are committed.

```
- (void)deleteObject:(NSManagedObject *)object
```

Parameters

object

A managed object.

Discussion

When changes are committed, *object* will be removed from the uniquing tables. If *object* has not yet been saved to a persistent store, it is simply removed from the receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [deletedObjects](#) (page 12)
- `isDeleted` (`NSManagedObject`)

Related Sample Code

Core Data HTML Store

CoreRecipes

Departments and Employees

QTMetadataEditor

Declared In

`NSManagedObjectContext.h`

detectConflictsForObject:

Marks an object for conflict detection.

```
- (void)detectConflictsForObject:(NSManagedObjectContext *)object
```

Parameters

object

A managed object.

Discussion

If on the next invocation of [save:](#) (page 26) *object* has been modified in its persistent store, the save fails. This allows optimistic locking for unchanged objects. Conflict detection is always performed on changed or deleted objects.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSManagedObjectContext.h

discardEditing

Causes the receiver to discard any changes in known editors, restoring the previous values

```
- (void)discardEditing
```

Discussion

Provides support for the `NSEditor` informal protocol. Causes the receiver to discard any changes in known editors, restoring the previous values. This method only applies to known editors (see [commitEditing](#) (page 10)). To discard general edits, use [rollback](#) (page 25) or [reset](#) (page 25).

See Also

- [commitEditing](#) (page 10)
- [commitEditingWithDelegate:didCommitSelector:contextInfo:](#) (page 11)
- [objectDidBeginEditing:](#) (page 18)
- [objectDidEndEditing:](#) (page 19)
- [reset](#) (page 25)
- [rollback](#) (page 25)

executeFetchRequest:error:

Returns an array of objects that meet the criteria specified by a given fetch request.

```
- (NSArray *)executeFetchRequest:(NSFetchRequest *)request error:(NSError **)error
```

Parameters

request

A fetch request that specifies the search criteria for the fetch.

error

If there is a problem executing the fetch, upon return contains an instance of `NSError` that describes the problem.

Return Value

An array of objects that meet the criteria specified by *request* fetched from the receiver and from the persistent stores associated with the receiver's persistent store coordinator. If an error occurs, returns `nil`. If no objects match the criteria specified by *request*, returns an empty array.

Discussion

Returned objects are registered with the receiver.

The following points are important to consider:

- If the fetch request has no predicate, then all instances of the specified entity are retrieved, modulo other criteria below.
- An object that meets the criteria specified by *request* (it is an instance of the entity specified by the request, and it matches the request's predicate if there is one) and that has been inserted into a context but which is not yet saved to a persistent store, is retrieved if the fetch request is executed on that context.
- If an object in a context has been modified, a predicate is evaluated against its modified state, not against the current state in the persistent store. Therefore, if an object in a context has been modified such that it meets the fetch request's criteria, the request retrieves it even if changes have not been saved to the store and the values in the store are such that it does not meet the criteria. Conversely, if an object in a context has been modified such that it does not match the fetch request, the fetch request will not retrieve it even if the version in the store does match.
- If an object has been deleted from the context, the fetch request does not retrieve it even if that deletion has not been saved to a store.

Objects that have been realized (populated, faults fired, “read from”, and so on) as well as pending updated, inserted, or deleted, are never changed by a fetch operation without developer intervention. If you fetch some objects, work with them, and then execute a new fetch that includes a superset of those objects, you do not get new instances or update data for the existing objects—you get the existing objects with their current in-memory state.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

Core Data HTML Store

CoreRecipes

Departments and Employees

QTMetadataEditor

Declared In

`NSManagedObjectContext.h`

hasChanges

Returns a Boolean value that indicates whether the receiver has uncommitted changes.

- (BOOL)hasChanges

Return Value

YES if the receiver has uncommitted changes, otherwise NO.

Discussion

This property is not key-value observing compliant (see *Key-Value Observing Programming Guide*)—if you are using Cocoa bindings, you cannot bind to the `hasChanges` property of a managed object context.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [save:](#) (page 26)
- [reset](#) (page 25)
- [rollback](#) (page 25)

Declared In

NSManagedObjectContext.h

insertedObjects

Returns the set of objects that have been inserted into the receiver but not yet saved in a persistent store.

```
- (NSSet *)insertedObjects
```

Return Value

The set of objects that have been inserted into the receiver but not yet saved in a persistent store.

Discussion

A managed object context does not post key-value observing notifications when the return value of `insertedObjects` changes—it does, however, post a [NSManagedObjectContextObjectsDidChangeNotification](#) (page 35) notification when a change is made, and a [NSManagedObjectContextDidSaveNotification](#) (page 35) notification when changes are committed.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [deletedObjects](#) (page 12)
- [insertObject:](#) (page 16)
- [registeredObjects](#) (page 24)
- [updatedObjects](#) (page 31)

Declared In

NSManagedObjectContext.h

insertObject:

Registers an object to be inserted in the receiver's persistent store the next time changes are saved.

```
- (void)insertObject:(NSManagedObject *)object
```

Parameters

object

A managed object.

Discussion

The managed object (*object*) is registered in the receiver with a temporary global ID. It is assigned a permanent global ID when changes are committed. If the current transaction is rolled back (for example, if the receiver is sent a [rollback](#) (page 25) message) before a save operation, the object is unregistered from the receiver.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [insertedObjects](#) (page 16)

Declared In

NSManagedObjectContext.h

lock

Attempts to acquire a lock on the receiver.

```
- (void)lock
```

Discussion

This method blocks a thread's execution until the lock can be acquired. An application protects a critical section of code by requiring a thread to acquire a lock before executing the code. Once the critical section is past, the thread relinquishes the lock by invoking [unlock](#) (page 31).

Sending this message to a managed object context helps the framework to understand the scope of a transaction in a multi-threaded environment. It is preferable to use the NSManagedObjectContext's implementation of NSLocking instead using of a separate mutex object.

If you [lock](#) (or successfully [tryLock](#)) a managed object context, the thread in which the lock call is made must have a retain until it invokes [unlock](#). If you do not properly retain a context in a multi-threaded environment, this will result in deadlock.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [tryLock](#) (page 30)

- [unlock](#) (page 31)

Declared In

NSManagedObjectContext.h

mergeChangesFromContextDidSaveNotification:

Merges the changes specified in a given notification.

```
- (void)mergeChangesFromContextDidSaveNotification:(NSNotification *)notification
```

Parameters*notification*

An instance of an [NSManagedObjectContextDidSaveNotification](#) (page 35) notification posted by another context.

Discussion

This method refreshes any objects which have been updated in the other context, faults in any newly-inserted objects, and invokes [deleteObject:](#) (page 13): on those which have been deleted.

You can use this method to, for example, update a managed object context on the main thread with work completed in another context in another thread. You must, though, [lock](#) (page 17) the receiver or otherwise ensure thread safety (that is, the notification contents are handled safely by Core Data, but the receiver's usage is still expected to conform to the standard Core Data threading policies). For example, you might implement a method to handle a notification that a worker thread had finished saving as follows:

```
- (void)workerThreadObjectContextDidSave:(NSNotification*)saveNotification {
    NSManagedObjectContext *appMOC = [[NSApp delegate] managedObjectContext];
    [appMOC
 performSelectorOnMainThread:@selector(mergeChangesFromContextDidSaveNotification:)
    withObject:saveNotification
    waitUntilDone:NO];
}
```

In this case, serialization is enforced by the main thread's run loop.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSManagedObjectContext.h

mergePolicy

Returns the merge policy of the receiver.

```
- (id)mergePolicy
```

Return Value

The receiver's merge policy.

Discussion

The default is `NSErrorMergePolicy`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSManagedObjectContext.h

objectDidBeginEditing:

Provides support for the `NSEditorRegistration` informal protocol.

```
- (void)objectDidBeginEditing:(id)editor
```

Parameters*editor*

An external editor that has changes that may affect the receiver.

Discussion

This message should be sent to the receiver when *editor* has uncommitted changes that can affect the receiver. There should typically be no reason for you to invoke this method directly.

See Also

- [commitEditing](#) (page 10)
- [commitEditingWithDelegate:didCommitSelector:contextInfo:](#) (page 11)
- [discardEditing](#) (page 14)
- [objectDidEndEditing:](#) (page 19)

objectDidEndEditing:

Provides support for the `NSEditorRegistration` informal protocol.

- (void)objectDidEndEditing:(id)*editor*

Parameters*editor*

An external editor that has made changes that affect the receiver.

Discussion

This message should be sent to the receiver when *editor* has finished editing a property belonging to the receiver. There should typically be no reason for you to invoke this method directly.

See Also

- [commitEditing](#) (page 10)
- [commitEditingWithDelegate:didCommitSelector:contextInfo:](#) (page 11)
- [discardEditing](#) (page 14)
- [objectDidBeginEditing:](#) (page 18)

objectRegisteredForID:

Returns the object for a specified ID, if the object is registered with the receiver.

- (NSManagedObject *)objectRegisteredForID:(NSManagedObjectID *)*objectID*

Parameters*objectID*

An object ID.

Return Value

The object for the specified ID if it is registered with the receiver, otherwise `nil`.

Availability

Available in Mac OS X v10.4 and later.

See Also

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

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

objectWithID:

Returns the object for a specified ID.

```
- (NSManagedObject *)objectWithID:(NSManagedObjectID *)objectID
```

Parameters*objectID*

An object ID.

Return Value

The object for the specified ID.

Discussion

If the object is not registered in the context, it may be fetched or returned as a fault. This method always returns an object. The data in the persistent store represented by *objectID* is assumed to exist—if it does not, the returned object throws an exception when you access any property (that is, when the fault is fired). The benefit of this behavior is that it allows you to create and use faults, then create the underlying rows later or in a separate context.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [objectRegisteredForID:](#) (page 19)
- `managedObjectIDForURIRepresentation:`
- `URIRepresentation`

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

observeValueForKeyPath:ofObject:change:context:

This message is sent to the receiver when the value at the specified key path relative to the given object has changed.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
  ofObject:(id)object
  change:(NSDictionary *)change
  context:(void *)context
```

Parameters*keyPath*The key path, relative to *object*, to the value that has changed.

object

The source object of the key path *keyPath*.

change

A dictionary that describes the changes that have been made to the value of the property at the key path *keyPath* relative to *object*. For possible values, see `NSKeyValueObserving`.

context

The value that was provided when the receiver was registered to receive key-value observation notifications.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSManagedObjectContext.h`

obtainPermanentIDsForObjects:error:

Converts to permanent IDs the object IDs of the objects in a given array.

```
- (BOOL)obtainPermanentIDsForObjects:(NSArray *)objects error:(NSError **)error
```

Parameters

objects

An array of managed objects.

error

If an error occurs, upon return contains an `NSError` object that describes the problem.

Return Value

YES if permanent IDs are obtained for all the objects in *objects*, otherwise NO.

Discussion

This method converts the object ID of each managed object in *objects* to a permanent ID. Although the object will have a permanent ID, it will still respond positively to `isInserted` until it is saved. Any object that already has a permanent ID is ignored.

Any object not already assigned to a store is assigned based on the same rules Core Data uses for assignment during a save operation (first writable store supporting the entity, and appropriate for the instance and its related items).

Special Considerations

This method results in a transaction with the underlying store which changes the file's modification date.

This results an additional consideration if you invoke this method on the managed object context associated with an instance of `NSPersistentDocument`. Instances of `NSDocument` need to know that they are in sync with the underlying content. To avoid problems, after invoking this method you must therefore update the document's modification date (using `setFileModificationDate:`).

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSManagedObjectContext.h`

persistentStoreCoordinator

Returns the persistent store coordinator of the receiver.

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
```

Return Value

The persistent store coordinator of the receiver.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

processPendingChanges

Forces the receiver to process changes to the object graph.

```
- (void)processPendingChanges
```

Discussion

This method causes changes to registered managed objects to be recorded with the undo manager.

In AppKit-based applications, this method is invoked automatically at least once during the event loop (at the end of the loop)—it may be called more often than that if the framework needs to coalesce your changes before doing something else. You can also invoke it manually to coalesce any pending unprocessed changes.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [redo](#) (page 23)
- [undo](#) (page 30)
- [undoManager](#) (page 30)

Related Sample Code

CoreRecipes

Departments and Employees

Declared In

NSManagedObjectContext.h

propagatesDeletesAtEndOfEvent

Returns a Boolean that indicates whether the receiver propagates deletes at the end of the event in which a change was made.

```
- (BOOL)propagatesDeletesAtEndOfEvent
```

Return Value

YES if the receiver propagates deletes at the end of the event in which a change was made, NO if it propagates deletes only immediately before saving changes.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setPropagatesDeletesAtEndOfEvent:](#) (page 27)

Declared In

NSManagedObjectContext.h

redo

Sends an redo message to the receiver's undo manager, asking it to reverse the latest undo operation applied to objects in the object graph.

- (void)redo

Availability

Available in Mac OS X v10.4 and later.

See Also

- [undo](#) (page 30)
- [processPendingChanges](#) (page 22)

Declared In

NSManagedObjectContext.h

refreshObject:mergeChanges:

Updates the persistent properties of a managed object to use the latest values from the persistent store.

- (void)refreshObject:(NSManagedObject *)*object* mergeChanges:(BOOL)*flag*

Parameters

object

A managed object.

flag

A Boolean value.

If *flag* is NO, then *object* is turned into a fault and any pending changes are lost. The object remains a fault until it is accessed again, at which time its property values will be reloaded from the store or last cached state.

If *flag* is YES, then *object's* property values are reloaded from the values from the store or the last cached state then any changes that were made (in the local context) are re-applied over those (now newly updated) values. (If *flag* is YES the merge of the values into *object* will always succeed—in this case there is therefore no such thing as a “merge conflict” or a merge that is not possible.)

Discussion

If the staleness interval (see [stalenessInterval](#) (page 29)) has not been exceeded, any available cached data is reused instead of executing a new fetch. If *flag* is YES, this method does not affect any transient properties; if *flag* is NO, transient properties are released.

You typically use this method to ensure data freshness if more than one managed object context may use the same persistent store simultaneously, in particular if you get an optimistic locking failure when attempting to save.

It is important to note that turning *object* into a fault (*flag* is NO) also causes related managed objects (that is, those to which *object* has a reference) to be released, so you can also use this method to trim a portion of your object graph you want to constrain memory usage.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [detectConflictsForObject](#): (page 14)
- [reset](#) (page 25)
- [setStalenessInterval](#): (page 28)

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

registeredObjects

Returns the set of objects registered with the receiver.

```
- (NSSet *)registeredObjects
```

Return Value

The set of objects registered with the receiver.

Discussion

A managed object context does not post key-value observing notifications when the return value of `registeredObjects` changes.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [deletedObjects](#) (page 12)
- [insertedObjects](#) (page 16)
- [updatedObjects](#) (page 31)

Declared In

NSManagedObjectContext.h

reset

Returns the receiver to its base state.

- (void)reset

Discussion

All the receiver's managed objects are “forgotten.” If you use this method, you should ensure that you also discard references to any managed objects fetched using the receiver, since they will be invalid afterwards.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [rollback](#) (page 25)
- [setStalenessInterval:](#) (page 28)
- [undo](#) (page 30)

Related Sample Code

QTMetadataEditor

Declared In

NSManagedObjectContext.h

retainsRegisteredObjects

Returns a Boolean that indicates whether the receiver sends a `retain` message to objects upon registration.

- (BOOL)retainsRegisteredObjects

Return Value

YES if the receiver sends a `retain` message to objects upon registration, otherwise NO.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setRetainsRegisteredObjects:](#) (page 28)

Declared In

NSManagedObjectContext.h

rollback

Removes everything from the undo stack, discards all insertions and deletions, and restores updated objects to their last committed values.

- (void)rollback

Discussion

This method does not refetch data from the persistent store or stores.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [reset](#) (page 25)
- [setStalenessInterval:](#) (page 28)
- [undo](#) (page 30)
- [processPendingChanges](#) (page 22)

Declared In

NSManagedObjectContext.h

save:

Attempts to commit unsaved changes to registered objects to their persistent store.

```
- (BOOL)save:(NSError **)error
```

Parameters

error

A pointer to an `NSError` object. You do not need to create an `NSError` object. The save operation aborts after the first failure if you pass `NULL`.

Return Value

YES if the save succeeds, otherwise NO.

Discussion

If there were multiple errors (for example several edited objects had validation failures) the description of `NSError` returned indicates that there were multiple errors, and its `userInfo` dictionary contains the key `NSDetailedErrors`. The value associated with the `NSDetailedErrors` key is an array that contains the individual `NSError` objects.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [reset](#) (page 25)
- [rollback](#) (page 25)
- [hasChanges](#) (page 15)

Related Sample Code

CoreRecipes

Declared In

NSManagedObjectContext.h

setMergePolicy:

Sets the merge policy of the receiver.

```
- (void)setMergePolicy:(id)mergePolicy
```

Parameters*mergePolicy*

The merge policy of the receiver. For possible values, see “Merge Policies” (page 33).

Availability

Available in Mac OS X v10.4 and later.

Declared In

NSManagedObjectContext.h

setPersistentStoreCoordinator:

Sets the persistent store coordinator of the receiver.

```
- (void)setPersistentStoreCoordinator:(NSPersistentStoreCoordinator *)coordinator
```

Parameters*coordinator*

The persistent store coordinator of the receiver.

Discussion

The coordinator provides the managed object model and handles persistency. Note that multiple contexts can share a coordinator.

This method raises an exception if *coordinator* is `nil`. If you want to “disconnect” a context from its persistent store coordinator, you should simply release all references to the context and allow it to be deallocated normally.

Availability

Available in Mac OS X v10.4 and later.

Related Sample Code

Core Data HTML Store

CoreRecipes

Declared In

NSManagedObjectContext.h

setPropagatesDeletesAtEndOfEvent:

Sets whether the context propagates deletes to related objects at the end of the event.

```
- (void)setPropagatesDeletesAtEndOfEvent:(BOOL)flag
```

Parameters*Flag*

A Boolean value that indicates whether the context propagates deletes to related objects at the end of the event (YES) or not (NO).

Discussion

The default is YES. If the value is NO, then deletes are propagated during a save operation.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [propagatesDeletesAtEndOfEvent](#) (page 22)

Declared In

NSManagedObjectContext.h

setRetainsRegisteredObjects:

Sets whether or not the receiver retains all registered objects, or only objects necessary for a pending save (those that are inserted, updated, deleted, or locked).

```
- (void)setRetainsRegisteredObjects:(BOOL)flag
```

Parameters

flag

A Boolean value.

If *flag* is NO, then registered objects are retained only when they are inserted, updated, deleted, or locked.

If *flag* is YES, then all registered objects are retained.

Discussion

The default is NO.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [retainsRegisteredObjects](#) (page 25)

Declared In

NSManagedObjectContext.h

setStalenessInterval:

Sets the staleness interval of the receiver.

```
- (void)setStalenessInterval:(NSTimeInterval)expiration
```

Parameters

expiration

The staleness interval of the receiver.

Discussion

The staleness interval controls whether *fulfilling a fault* uses data previously fetched by the application, or issues a new fetch (see also [refreshObject:mergeChanges:](#) (page 23)). The staleness interval does *not* affect objects currently in use (that is, it is *not* used to automatically update property values from a persistent store after a certain period of time).

The expiration value is applied on a per object basis. It is the relative time until cached data (snapshots) should be considered stale. For example, a value of 300.0 informs the context to utilize cached information for no more than 5 minutes after an object was originally fetched.

The default is infinite staleness (represented by an interval of 0).

Note that the staleness interval is a hint and may not be supported by all persistent store types. It is not used by XML and binary stores, since these stores maintain all current values in memory.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [reset](#) (page 25)
- [rollback](#) (page 25)
- [stalenessInterval](#) (page 29)
- [undo](#) (page 30)
- [refreshObject:mergeChanges:](#) (page 23)

Declared In

NSManagedObjectContext.h

setUndoManager:

Sets the undo manager of the receiver.

```
- (void)setUndoManager:(NSUndoManager *)undoManager
```

Parameters

undoManager

The undo manager of the receiver.

Discussion

By default, a context provides its own undo manager. You can set the undo manager to `nil` to disable undo support, for example in a large import process. For more details, see *Core Data Programming Guide*.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [undoManager](#) (page 30)

Declared In

NSManagedObjectContext.h

stalenessInterval

Returns the staleness interval of the receiver.

```
- (NSTimeInterval)stalenessInterval
```

Return Value

The staleness interval of the receiver.

Discussion

For more details, see [setStalenessInterval:](#) (page 28).

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setStalenessInterval:](#) (page 28)

Declared In

NSManagedObjectContext.h

tryLock

Attempts to acquire a lock.

- (BOOL)tryLock

Return Value

YES if a lock was acquired, NO otherwise.

Discussion

This method returns immediately after the attempt to acquire a lock.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [lock](#) (page 17)

- [unlock](#) (page 31)

Declared In

NSManagedObjectContext.h

undo

Sends an undo message to the receiver's undo manager, asking it to reverse the latest uncommitted changes applied to objects in the object graph.

- (void)undo

Availability

Available in Mac OS X v10.4 and later.

See Also

- [reset](#) (page 25)

- [rollback](#) (page 25)

- [undoManager](#) (page 30)

- [processPendingChanges](#) (page 22)

Declared In

NSManagedObjectContext.h

undoManager

Returns the undo manager of the receiver.

- (NSUndoManager *)undoManager

Return Value

The undo manager of the receiver.

Discussion

By default, a context provides its own undo manager. For more details, see [setUndoManager:](#) (page 29).

Availability

Available in Mac OS X v10.4 and later.

See Also

- [setUndoManager:](#) (page 29)

Related Sample Code

Departments and Employees

Declared In

NSManagedObjectContext.h

unlock

Relinquishes a previously acquired lock.

- (void)unlock

Availability

Available in Mac OS X v10.4 and later.

See Also

- [lock](#) (page 17)
- [tryLock](#) (page 30)

Declared In

NSManagedObjectContext.h

updatedObjects

Returns the set of objects registered with the receiver that have uncommitted changes.

- (NSSet *)updatedObjects

Return Value

The set of objects registered with the receiver that have uncommitted changes.

Discussion

A managed object context does not post key-value observing notifications when the return value of `updatedObjects` changes—it does, however, post a [NSManagedObjectContextObjectsDidChangeNotification](#) (page 35) notification when a change is made, and a [NSManagedObjectContextDidSaveNotification](#) (page 35) notification when changes are committed.

Availability

Available in Mac OS X v10.4 and later.

See Also

- [deletedObjects](#) (page 12)
- [insertedObjects](#) (page 16)
- [registeredObjects](#) (page 24)

Declared In

NSManagedObjectContext.h

Constants

NSManagedObjectContext Change Notification User Info Keys

Core Data uses these string constants as keys in the user info dictionary in managed object context notifications ([NSManagedObjectContextObjectsDidChangeNotification](#) (page 35) and [NSManagedObjectContextDidSaveNotification](#) (page 35)).

```
NSString * const NSInsertedObjectsKey;
NSString * const NSUpdatedObjectsKey;
NSString * const NSDeletedObjectsKey;
NSString * const NSRefreshedObjectsKey;
NSString * const NSInvalidatedObjectsKey;
NSString * const NSInvalidatedAllObjectsKey;
```

Constants

NSInsertedObjectsKey

Key for the set of objects that were inserted into the context.

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

NSUpdatedObjectsKey

Key for the set of objects that were updated.

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

NSDeletedObjectsKey

Key for the set of objects that were marked for deletion during the previous event.

Note that the set of deleted objects given for a [NSManagedObjectContextDidSaveNotification](#) (page 35) does not include objects that were inserted and deleted without an intervening save operation—that is, they had not been saved to a persistent store. See also [deletedObjects](#) (page 12) (NSManagedObjectContext) and [isDeleted](#) (NSManagedObject).

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

NSRefreshedObjectsKey

Key for the set of objects that were refreshed.

Available in Mac OS X v10.5 and later.

Declared in NSManagedObjectContext.h.

NSInvalidatedObjectsKey

Key for the set of objects that were invalidated.

Available in Mac OS X v10.5 and later.

Declared in NSManagedObjectContext.h.

NSInvalidatedAllObjectsKey

Key that specifies that all objects in the context have been invalidated.

Available in Mac OS X v10.5 and later.

Declared in NSManagedObjectContext.h.

Declared In

NSManagedObjectContext.h

Merge Policies

Merge policy constants define the way conflicts are handled during a save operation.

```
id NSErrorMergePolicy;
id NSMergeByPropertyStoreTrumpMergePolicy;
id NSMergeByPropertyObjectTrumpMergePolicy;
id NSOverwriteMergePolicy;
id NSRollbackMergePolicy;
```

Constants

NSErrorMergePolicy

This policy causes a save to fail if there are any merge conflicts.

In the case of failure, the save method returns with an error with a userInfo dictionary that contains the key @"conflictList"; the corresponding value is an array of conflict records.

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

NSMergeByPropertyStoreTrumpMergePolicy

This policy merges conflicts between the persistent store's version of the object and the current in-memory version, giving priority to external changes.

The merge occurs by individual property. For properties that have been changed in both the external source and in memory, the external changes trump the in-memory ones.

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

NSMergeByPropertyObjectTrumpMergePolicy

This policy merges conflicts between the persistent store's version of the object and the current in-memory version, giving priority to in-memory changes.

The merge occurs by individual property. For properties that have been changed in both the external source and in memory, the in-memory changes trump the external ones.

Available in Mac OS X v10.4 and later.

Declared in NSManagedObjectContext.h.

`NSOverwriteMergePolicy`

This policy overwrites state in the persistent store for the changed objects in conflict.

Changed objects' current state is forced upon the persistent store.

Available in Mac OS X v10.4 and later.

Declared in `NSManagedObjectContext.h`.

`NSRollbackMergePolicy`

This policy discards in-memory state changes for objects in conflict.

The persistent store's version of the objects' state is used.

Available in Mac OS X v10.4 and later.

Declared in `NSManagedObjectContext.h`.

Discussion

The default policy is the `NSErrorMergePolicy`. It is the only policy that requires action to correct any conflicts; the other policies make a save go through silently by making changes following their rules.

Declared In

`NSManagedObjectContext.h`

The following constants, defined in `CoreDataErrors.h`, relate to errors returned following validation failures or problems encountered during a save operation.

<code>NSValidationObjectErrorKey</code>	Key for the object that failed to validate for a validation error.
<code>NSAffectedStoresErrorKey</code>	The key for stores prompting an error.
<code>NSAffectedObjectsErrorKey</code>	The key for objects prompting an error.

Each conflict record in the `@"conflictList"` array in the `userInfo` dictionary for an error from the `NSErrorMergePolicy` is a dictionary containing some of the keys described in the following table. Of the `cachedRow`, `databaseRow`, and `snapshot` keys, only two will be present depending on whether the conflict is between the managed object context and the persistent store coordinator (`snapshot` and `cachedRow`) or between the persistent store coordinator and the persistent store (`cachedRow` and `databaseRow`).

Constant	Description
<code>@"object"</code>	The managed object that could not be saved.
<code>@"snapshot"</code>	A dictionary of key-value pairs for the properties that represents the managed object context's last saved state for this managed object.
<code>@"cachedRow"</code>	A dictionary of key-value pairs for the properties that represents the persistent store's last saved state for this managed object.
<code>@"databaseRow"</code>	A dictionary of key-value pairs for the properties that represents the database's current state for this managed object.
<code>@"newVersion"</code>	An <code>NSNumber</code> object whose value is latest version number of this managed object.
<code>@"oldVersion"</code>	As <code>NSNumber</code> object whose value is the version number that this managed object context last saved for this managed object.

Notifications

NSManagedObjectContextObjectsDidChangeNotification

Posted when values of properties of objects contained in a managed object context are changed.

The notification is posted during [processPendingChanges](#) (page 22), after the changes have been processed, but before it is safe to call [save](#): (page 26) again (if you try, you will generate an infinite loop).

The notification object is the managed object context. The *userInfo* dictionary contains the following keys: `NSInsertedObjectsKey`, `NSUpdatedObjectsKey`, and `NSDeletedObjectsKey`.

Note that this notification is posted only when managed objects are *changed*; it is not posted when managed objects are added to a context as the result of a fetch.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSManagedObjectContext.h`

NSManagedObjectContextDidSaveNotification

Posted whenever a managed object context completes a save operation.

The notification object is the managed object context. The *userInfo* dictionary contains the following keys: `NSInsertedObjectsKey`, `NSUpdatedObjectsKey`, and `NSDeletedObjectsKey`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`NSManagedObjectContext.h`

Document Revision History

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

Date	Notes
2008-10-15	Clarified return value for <code>countForFetchRequest:error:</code> .
2007-07-19	Updated for Mac OS X v10.5.
2007-03-06	Clarified description of <code>NSManagedObjectContextObjectsDidChangeNotification</code> .
2007-01-08	Noted that <code>hasChanges</code> is not KVO-compliant, and enhanced discussion of <code>setPersistentStoreCoordinator:</code> .
2006-11-07	Clarified effect of <code>refreshObject:mergeChanges:</code> .
2006-05-23	First publication of this content as a separate document.

REVISION HISTORY

Document Revision History

Index

Symbols

@"cachedRow" constant [34](#)
@"databaseRow" constant [34](#)
@"newVersion" constant [34](#)
@"object" constant [34](#)
@"oldVersion" constant [34](#)
@"snapshot" constant [34](#)

A

assignObject:toPersistentStore: instance method [9](#)

C

commitEditing instance method [10](#)
commitEditingWithDelegate:didCommitSelector:
contextInfo: instance method [11](#)
countForFetchRequest:error: instance method [12](#)

D

deletedObjects instance method [12](#)
deleteObject: instance method [13](#)
detectConflictsForObject: instance method [14](#)
discardEditing instance method [14](#)

E

executeFetchRequest:error: instance method [14](#)

H

hasChanges instance method [15](#)

I

insertedObjects instance method [16](#)
insertObject: instance method [16](#)

L

lock instance method [17](#)

M

Merge Policies [33](#)
mergeChangesFromContextDidSaveNotification:
instance method [17](#)
mergePolicy instance method [18](#)

N

NSDeletedObjectsKey constant [32](#)
NSErrorMergePolicy constant [33](#)
NSInsertedObjectsKey constant [32](#)
NSInvalidatedAllObjectsKey constant [33](#)
NSInvalidatedObjectsKey constant [33](#)
NSManagedObjectContext Change Notification User Info
Keys [32](#)
NSManagedObjectContextDidSaveNotification
notification [35](#)
NSManagedObjectContextObjectsDidChangeNotification
notification [35](#)
NSMergeByPropertyObjectTrumpMergePolicy
constant [33](#)

NSMergeByPropertyStoreTrumpMergePolicy
constant 33
 NSOverwriteMergePolicy **constant 34**
 NSRefreshedObjectsKey **constant 33**
 NSRollbackMergePolicy **constant 34**
 NSUpdatedObjectsKey **constant 32**

O

objectDidBeginEditing: **instance method 18**
 objectDidEndEditing: **instance method 19**
 objectRegisteredForID: **instance method 19**
 objectWithID: **instance method 20**
 observeValueForKeyPath:ofObject:change:context:
instance method 20
 obtainPermanentIDsForObjects:error: **instance
 method 21**

P

persistentStoreCoordinator **instance method 22**
 processPendingChanges **instance method 22**
 propagatesDeletesAtEndOfEvent **instance method
 22**

R

redo **instance method 23**
 refreshObject:mergeChanges: **instance method 23**
 registeredObjects **instance method 24**
 reset **instance method 25**
 retainsRegisteredObjects **instance method 25**
 rollback **instance method 25**

S

save: **instance method 26**
 setMergePolicy: **instance method 26**
 setPersistentStoreCoordinator: **instance method
 27**
 setPropagatesDeletesAtEndOfEvent: **instance
 method 27**
 setRetainsRegisteredObjects: **instance method 28**
 setStalenessInterval: **instance method 28**
 setUndoManager: **instance method 29**
 stalenessInterval **instance method 29**

T

tryLock **instance method 30**

U

undo **instance method 30**
 undoManager **instance method 30**
 unlock **instance method 31**
 updatedObjects **instance method 31**