
NSPersistentStoreCoordinator Class Reference

Data Management



2009-05-01



Apple Inc.
© 2009 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, Mac OS, and Spotlight 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

NSPersistentStoreCoordinator Class Reference 5

Overview	5
Tasks	6
Registered Store Types	6
Initializing a Coordinator	6
Configuring Persistent Stores	6
Locking	6
Working with Metadata	7
Discovering Object IDs	7
Class Methods	7
metadataForPersistentStoreOfType:URL:error:	7
registeredStoreTypes	8
registerStoreClass:forStoreType:	8
setMetadata:forPersistentStoreOfType:URL:error:	9
Instance Methods	9
addPersistentStoreWithType:configuration:URL:options:error:	9
initWithManagedObjectModel:	10
lock	11
managedObjectIDForURIRepresentation:	11
managedManagedObjectModel	11
metadataForPersistentStore:	12
migratePersistentStore:toURL:options:withType:error:	12
persistentStoreForURL:	13
persistentStores	14
removePersistentStore:error:	14
setMetadata:forPersistentStore:	14
setURL:forPersistentStore:	15
tryLock	16
unlock	16
URLForPersistentStore:	16
Constants	17
Store Types	17
Store Metadata	17
Stores Change Notification User Info Keys	18
Store Options	18
Migration Options	19
Versioning Support	20
Notifications	21
NSPersistentStoreCoordinatorStoresDidChangeNotification	21
NSPersistentStoreCoordinatorWillRemoveStoreNotification	21

Document Revision History 23

NSPersistentStoreCoordinator Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.framework
Availability	Available in iOS 3.0 and later.
Declared in	NSPersistentStoreCoordinator.h
Companion guides	Core Data Programming Guide Atomic Store Programming Topics Core Data Spotlight Integration Programming Guide

Overview

Instances of `NSPersistentStoreCoordinator` associate persistent stores (by type) with a model (or more accurately, a configuration of a model) and serve to mediate between the persistent store or stores and the managed object context or contexts. Instances of `NSManagedObjectContext` use a coordinator to save object graphs to persistent storage and to retrieve model information. A context without a coordinator is not fully functional as it cannot access a model except through a coordinator. The coordinator is designed to present a façade to the managed object contexts such that a group of persistent stores appears as an aggregate store. A managed object context can then create an object graph based on the union of all the data stores the coordinator covers.

Coordinators do the opposite of providing for concurrency—they **do not** specialize operations. If you want to use multiple threads for different write operations you use multiple coordinators. Note that if multiple threads work directly with a coordinator, they need to lock and unlock it explicitly.

Each coordinator (and thus container) may use different copies, and hence different versions, of a managed object model. This allows you to cleanly deal with file versioning.

The coordinator gives access to its underlying object stores. You can retrieve an object store when you first add one (using `addPersistentStoreWithType:configuration:URL:options:error:` (page 9)), or by using `persistentStoreForURL:` (page 13) or `persistentStores` (page 14). This allows you to determine, for example, whether a store has already been added, or whether two objects come from the same store.

- You move a store from one location to another, or change the type of a store, using `migratePersistentStore:toURL:options:withType:error:` (page 12).
- You can set metadata for a given store using the persistent store coordinator (`setMetadata:forPersistentStore:` (page 14)).

For more details about these tasks, see Persistent Store Features in *Core Data Programming Guide*.

Tasks

Registered Store Types

- + [registeredStoreTypes](#) (page 8)
Returns a dictionary of the registered store types.
- + [registerStoreClass:forStoreType:](#) (page 8)
Registers a given `NSPersistentStore` subclass for a given store type string.

Initializing a Coordinator

- [initWithManagedObjectModel:](#) (page 10)
Initializes the receiver with a managed object model.
- [managedObjectModel](#) (page 11)
Returns the receiver's managed object model.

Configuring Persistent Stores

- [addPersistentStoreWithType:configuration:URL:options:error:](#) (page 9)
Adds a new persistent store of a specified type at a given location, and returns the new store.
- [setURL:forPersistentStore:](#) (page 15)
Sets the URL for a given persistent store.
- [removePersistentStore:error:](#) (page 14)
Removes a given persistent store.
- [migratePersistentStore:toURL:options:withType:error:](#) (page 12)
Moves a persistent store to a new location, changing the storage type if necessary.
- [persistentStores](#) (page 14)
Returns an array of persistent stores associated with the receiver.
- [persistentStoreForURL:](#) (page 13)
Returns the persistent store for the specified URL.
- [URLForPersistentStore:](#) (page 16)
Returns the URL for a given persistent store.

Locking

- [lock](#) (page 11)
Attempts to acquire a lock.
- [tryLock](#) (page 16)
Attempts to acquire a lock.

- [unlock](#) (page 16)
Relinquishes a previously acquired lock.

Working with Metadata

- [metadataForPersistentStore:](#) (page 12)
Returns a dictionary that contains the metadata currently stored or to-be-stored in a given persistent store.
- [setMetadata:forPersistentStore:](#) (page 14)
Sets the metadata stored in the persistent store during the next save operation executed on it to *metadata*.
- + [setMetadata:forPersistentStoreOfType:URL:error:](#) (page 9)
Sets the metadata for a given store.
- + [metadataForPersistentStoreOfType:URL:error:](#) (page 7)
Returns a dictionary containing the metadata stored in the persistent store at a given URL.

Discovering Object IDs

- [managedObjectIDForURIRepresentation:](#) (page 11)
Returns an object ID for the specified URI representation of an object ID if a matching store is available, or *nil* if a matching store cannot be found.

Class Methods

metadataForPersistentStoreOfType:URL:error:

Returns a dictionary containing the metadata stored in the persistent store at a given URL.

```
+ (NSDictionary *)metadataForPersistentStoreOfType:(NSString *)storeType URL:(NSURL *)url error:(NSError **)error
```

Parameters

storeType

The type of the store at *url*. If this value is *nil*, Core Data determines which store class should be used to get or set the store file's metadata by inspecting the file contents.

url

The location of a persistent store.

error

If no store is found at *url* or if there is a problem accessing its contents, upon return contains an `NSError` object that describes the problem.

Return Value

A dictionary containing the metadata stored in the persistent store at *url*, or *nil* if the store cannot be opened or if there is a problem accessing its contents.

The keys guaranteed to be in this dictionary are [NSToreTypeKey](#) (page 17) and [NSStoreUUIDKey](#) (page 18).

Discussion

You can use this method to retrieve the metadata from a store without the overhead of creating a Core Data stack.

Availability

Available in iOS 3.0 and later.

See Also

- + [setMetadata:forPersistentStoreOfURL:error:](#) (page 9)
- [metadataForPersistentStore:](#) (page 12)
- [setMetadata:forPersistentStore:](#) (page 14)

Declared In

NSPersistentStoreCoordinator.h

registeredStoreTypes

Returns a dictionary of the registered store types.

```
+ (NSDictionary *)registeredStoreTypes
```

Return Value

A dictionary of the registered store types—the keys are the store type strings, and the values are the `NSPersistentStore` subclasses.

Availability

Available in iOS 3.0 and later.

Declared In

NSPersistentStoreCoordinator.h

registerStoreClass:forStoreType:

Registers a given `NSPersistentStore` subclass for a given store type string.

```
+ (void)registerStoreClass:(Class)storeClass forStoreType:(NSString *)storeType
```

Parameters

storeClass

The `NSPersistentStore` subclass to use for the store of type *storeType*.

storeType

A unique string that identifies a store type.

Discussion

You must invoke this method before a custom subclass of `NSPersistentStore` can be loaded into a persistent store coordinator.

You can pass `nil` for *storeClass* to unregister the store type.

Availability

Available in iOS 3.0 and later.

Declared In

NSPersistentStoreCoordinator.h

setMetadata:forPersistentStoreOfType:URL:error:

Sets the metadata for a given store.

```
+ (BOOL)setMetadata:(NSDictionary *)metadata forPersistentStoreOfType:(NSString *)storeType URL:(NSURL *)url error:(NSError **)error
```

Parameters

metadata

A dictionary containing metadata for the store.

storeType

The type of the store at *url*. If this value is *nil*, Core Data will determine which store class should be used to get or set the store file's metadata by inspecting the file contents.

url

The location of a persistent store.

error

If no store is found at *url* or if there is a problem setting its metadata, upon return contains an `NSError` object that describes the problem.

Return Value

YES if the metadata was set correctly, otherwise NO.

Discussion

You can use this method to set the metadata for a store without the overhead of creating a Core Data stack.

Availability

Available in iOS 3.0 and later.

See Also

- + [metadataForPersistentStoreOfType:URL:error:](#) (page 7)
- [metadataForPersistentStore:](#) (page 12)
- [setMetadata:forPersistentStore:](#) (page 14)

Declared In

NSPersistentStoreCoordinator.h

Instance Methods

addPersistentStoreWithType:configuration:URL:options:error:

Adds a new persistent store of a specified type at a given location, and returns the new store.

```
- (NSPersistentStore *)addPersistentStoreWithType:(NSString *)storeType
    configuration:(NSString *)configuration URL:(NSURL *)storeURL
    options:(NSDictionary *)options error:(NSError **)error
```

Parameters*storeType*

A string constant (such as `NSSQLiteStoreType`) that specifies the store type—see [“Store Types”](#) (page 17) for possible values.

configuration

The name of a configuration in the receiver's managed object model that will be used by the new store. The configuration can be `nil`, in which case no other configurations are allowed.

storeURL

The file location of the persistent store.

options

A dictionary containing key-value pairs that specify whether the store should be read-only, and whether (for an XML store) the XML file should be validated against the DTD before it is read. For key definitions, see [“Store Options”](#) (page 18) and [“Migration Options”](#) (page 19). This value may be `nil`.

error

If a new store cannot be created, upon return contains an instance of `NSError` that describes the problem

Return Value

The newly-created store or, if an error occurs, `nil`.

Availability

Available in iOS 3.0 and later.

See Also

- [migratePersistentStore:toURL:options:withType:error:](#) (page 12)
- [removePersistentStore:error:](#) (page 14)

Declared In

`NSPersistentStoreCoordinator.h`

initWithManagedObjectModel:

Initializes the receiver with a managed object model.

```
- (id)initWithManagedObjectModel:(NSManagedObjectModel *)model
```

Parameters*model*

A managed object model.

Return Value

The receiver, initialized with *model*.

Availability

Available in iOS 3.0 and later.

Declared In

`NSPersistentStoreCoordinator.h`

lock

Attempts to acquire a lock.

- (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`.

Availability

Available in iOS 3.0 and later.

See Also

- [tryLock](#) (page 16)
- [unlock](#) (page 16)

Declared In

NSPersistentStoreCoordinator.h

managedObjectIDForURIRepresentation:

Returns an object ID for the specified URI representation of an object ID if a matching store is available, or `nil` if a matching store cannot be found.

- (NSManagedObjectID *)managedObjectIDForURIRepresentation:(NSURL *)URL

Parameters

URL

An URL object containing a URI that specify a managed object.

Return Value

An object ID for the object specified by *URL*.

Discussion

The URI representation contains a UUID of the store the ID is coming from, and the coordinator can match it against the stores added to it.

Availability

Available in iOS 3.0 and later.

See Also

`URIRepresentation` (NSManagedObjectID)
`objectWithID:` (NSManagedObjectContext)

Declared In

NSPersistentStoreCoordinator.h

managedObjectModel

Returns the receiver's managed object model.

- (NSManagedObjectModel *)managedObjectModel

Return Value

The receiver's managed object model.

Availability

Available in iOS 3.0 and later.

Declared In

NSPersistentStoreCoordinator.h

metadataForPersistentStore:

Returns a dictionary that contains the metadata currently stored or to-be-stored in a given persistent store.

```
- (NSDictionary *)metadataForPersistentStore:(NSPersistentStore *)store
```

Parameters

store

A persistent store.

Return Value

A dictionary that contains the metadata currently stored or to-be-stored in *store*.

Availability

Available in iOS 3.0 and later.

See Also

- [setMetadata:forPersistentStore:](#) (page 14)
- + [metadataForPersistentStoreOfType:URL:error:](#) (page 7)
- + [setMetadata:forPersistentStoreOfType:URL:error:](#) (page 9)

Declared In

NSPersistentStoreCoordinator.h

migratePersistentStore:toURL:options:withType:error:

Moves a persistent store to a new location, changing the storage type if necessary.

```
- (NSPersistentStore *)migratePersistentStore:(NSPersistentStore *)store toURL:(NSURL *)URL options:(NSDictionary *)options withType:(NSString *)storeType error:(NSError **)error
```

Parameters

store

A persistent store.

URL

An URL object that specifies the location for the new store.

options

A dictionary containing key-value pairs that specify whether the store should be read-only, and whether (for an XML store) the XML file should be validated against the DTD before it is read. For key definitions, see ["Store Options"](#) (page 18).

storeType

A string constant (such as `NSSQLiteStoreType`) that specifies the type of the new store—see “[Store Types](#)” (page 17).

error

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

Return Value

If the migration is successful, the new store, otherwise `nil`.

Discussion

This method is typically used for “Save As” operations. Performance may vary depending on the type of old and new store. For more details of the action of this method, see [Persistent Store Features in Core Data Programming Guide](#).

Important: After invocation of this method, the specified store is removed from the coordinator thus `store` is no longer a useful reference.

Availability

Available in iOS 3.0 and later.

See Also

- [addPersistentStoreWithType:configuration:URL:options:error:](#) (page 9)
- [removePersistentStore:error:](#) (page 14)

Declared In

`NSPersistentStoreCoordinator.h`

persistentStoreForURL:

Returns the persistent store for the specified URL.

```
- (NSPersistentStore *)persistentStoreForURL:(NSURL *)URL
```

Parameters

URL

An URL object that specifies the location of a persistent store.

Return Value

The persistent store at the location specified by *URL*.

Availability

Available in iOS 3.0 and later.

See Also

- [persistentStores](#) (page 14)
- [URLForPersistentStore:](#) (page 16)

Declared In

`NSPersistentStoreCoordinator.h`

persistentStores

Returns an array of persistent stores associated with the receiver.

- (NSArray *)persistentStores

Return Value

An array persistent stores associated with the receiver.

Availability

Available in iOS 3.0 and later.

See Also

- [persistentStoreForURL:](#) (page 13)
- [URLForPersistentStore:](#) (page 16)

Declared In

NSPersistentStoreCoordinator.h

removePersistentStore:error:

Removes a given persistent store.

- (BOOL)removePersistentStore:(NSPersistentStore *)store error:(NSError **)error

Parameters

store

A persistent store.

error

If an error occurs, upon return contains an instance of NSError that describes the problem.

Return Value

YES if the store is removed, otherwise NO.

Availability

Available in iOS 3.0 and later.

See Also

- [addPersistentStoreWithType:configuration:URL:options:error:](#) (page 9)
- [migratePersistentStore:toURL:options:withType:error:](#) (page 12)

Declared In

NSPersistentStoreCoordinator.h

setMetadata:forPersistentStore:

Sets the metadata stored in the persistent store during the next save operation executed on it to *metadata*.

- (void)setMetadata:(NSDictionary *)metadata forPersistentStore:(NSPersistentStore *)store

Parameters*metadata*

A dictionary containing metadata for the store.

store

A persistent store.

Discussion

The store type and UUID (`NSStoreTypeKey` and `NSStoreUUIDKey`) are always added automatically, however `NSStoreUUIDKey` is only added if it is not set manually as part of the dictionary argument.

Important: Setting the metadata for a store does not change the information on disk until the store is actually saved.

Availability

Available in iOS 3.0 and later.

See Also

- [metadataForPersistentStore:](#) (page 12)
- + [setMetadata:forPersistentStoreOfType:URL:error:](#) (page 9)
- + [metadataForPersistentStoreOfType:URL:error:](#) (page 7)

Declared In

NSPersistentStoreCoordinator.h

setURL:forPersistentStore:

Sets the URL for a given persistent store.

```
-(BOOL)setURL:(NSURL *)url forPersistentStore:(NSPersistentStore *)store
```

Parameters*url*The new location for *store*.*store*

A persistent store associated with the receiver.

Return Value

YES if the store was relocated, otherwise NO.

Discussion

For atomic stores, this method alters the location to which the next save operation will write the file; for non-atomic stores, invoking this method will release the existing connection and create a new one at the specified URL. (For non-atomic stores, a store must already exist at the destination URL; a new store will not be created.)

Availability

Available in iOS 3.0 and later.

Declared In

NSPersistentStoreCoordinator.h

tryLock

Attempts to acquire a lock.

- (BOOL)tryLock

Return Value

YES if successful, otherwise NO.

Discussion

Returns immediately—contrast [lock](#) (page 11) which blocks.

Availability

Available in iOS 3.0 and later.

See Also

- [lock](#) (page 11)
- [unlock](#) (page 16)

Declared In

NSPersistentStoreCoordinator.h

unlock

Relinquishes a previously acquired lock.

- (void)unlock

Availability

Available in iOS 3.0 and later.

See Also

- [lock](#) (page 11)
- [tryLock](#) (page 16)

Declared In

NSPersistentStoreCoordinator.h

URLForPersistentStore:

Returns the URL for a given persistent store.

- (NSURL *)URLForPersistentStore:(NSPersistentStore *)store

Parameters

store

A persistent store.

Return Value

The URL for *store*.

Availability

Available in iOS 3.0 and later.

See Also

- [persistentStoreForURL:](#) (page 13)
- [persistentStores](#) (page 14)

Declared In

NSPersistentStoreCoordinator.h

Constants

Store Types

Types of persistent store.

```
NSString * const NSSQLiteStoreType;
NSString * const NSBinaryStoreType;
NSString * const NSInMemoryStoreType;
```

Constants

NSSQLiteStoreType

The SQLite database store type.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

NSBinaryStoreType

The binary store type.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

NSInMemoryStoreType

The in-memory store type.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Store Metadata

Keys used in a store's metadata dictionary.

```
NSString * const NSStoreTypeKey;
NSString * const NSStoreUUIDKey;
```

Constants

NSStoreTypeKey

The key in the metadata dictionary to identify the store type.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

NSStoreUUIDKey

The key in the metadata dictionary to identify the store UUID.

The store UUID is useful to identify stores through URI representations, but it is *not* guaranteed to be unique. The UUID generated for new stores is unique—users can freely copy files and thus the UUID stored inside—so if you track or reference stores explicitly you need to be aware of duplicate UUIDs and potentially override the UUID when a new store is added to the list of known stores in your application.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Declared In

`NSPersistentStoreCoordinator.h`

Stores Change Notification User Info Keys

An `NSPersistentStoreCoordinatorStoresDidChangeNotification` (page 21) notification is posted whenever persistent stores are added to or removed from a persistent store coordinator, or when store UUIDs change. The `userInfo` dictionary contains information about the stores that were added or removed using these keys.

```
NSString * const NSAddedPersistentStoresKey;
NSString * const NSRemovedPersistentStoresKey;
NSString * const NSUUIDChangedPersistentStoresKey;
```

Constants

`NSAddedPersistentStoresKey`

Key for the array of stores that were added.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSRemovedPersistentStoresKey`

Key for the array of stores that were removed.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSUUIDChangedPersistentStoresKey`

Key for the array of stores whose UUIDs changed.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Declared In

`NSPersistentStoreCoordinator.h`

Store Options

Keys for the options dictionary used in

`addPersistentStoreWithType:configuration:URL:options:error:` (page 9),

`migratePersistentStore:toURL:options:withType:error:` (page 12), and

`importStoreWithIdentifier:fromExternalRecordsDirectory:toURL:options:withType:error:`.

```

NSString * const NSReadOnlyPersistentStoreOption;
NSString * const NSPersistentStoreTimeoutOption;
NSString * const NSSQLitePragmasOption;
NSString * const NSSQLiteAnalyzeOption;
NSString * const NSSQLiteManualVacuumOption;

```

Constants

`NSReadOnlyPersistentStoreOption`

A flag that indicates whether a store is treated as read-only or not.

The default value is NO.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSPersistentStoreTimeoutOption`

Options key that specifies the connection timeout for Core Data stores.

The corresponding value is an `NSNumber` object that represents the duration in seconds that Core Data will wait while attempting to create a connection to a persistent store. If a connection is cannot be made within that timeframe, the operation is aborted and an error is returned.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSSQLitePragmasOption`

Options key for a dictionary of SQLite pragma settings with pragma values indexed by pragma names as keys.

All pragma values must be specified as `NSString` objects. The `fullfsync` and `synchronous` pragmas control the tradeoff between write performance (write to disk speed & cache utilization) and durability (data loss/corruption sensitivity to power interruption). For more information on pragma settings, see <http://sqlite.org/pragma.html>.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSSQLiteAnalyzeOption`

Option key to run an analysis of the store data to optimize indices based on statistical information when the store is added to the coordinator.

This invokes SQLite's `ANALYZE` command. It is ignored by stores other than the SQLite store.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSSQLiteManualVacuumOption`

Option key to rebuild the store file, forcing a database wide defragmentation when the store is added to the coordinator.

This invokes SQLite's `VACUUM` command. It is ignored by stores other than the SQLite store.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Migration Options

Migration options, specified in the dictionary of options when adding a persistent store using `addPersistentStoreWithType:configuration:URL:options:error:` (page 9).

```
NSString * const NSIgnorePersistentStoreVersioningOption;
NSString * const NSMigratePersistentStoresAutomaticallyOption;
NSString * const NSInferMappingModelAutomaticallyOption;
```

Constants

`NSIgnorePersistentStoreVersioningOption`

Key to ignore the built-in versioning provided by Core Data.

The corresponding value is an `NSNumber` object. If the `boolValue` of the number is `YES`, Core Data will not compare the version hashes between the managed object model in the coordinator and the metadata for the loaded store. (It will, however, continue to update the version hash information in the metadata.) This key and corresponding value of `YES` is specified by default for all applications linked on or before Mac OS X 10.4.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSMigratePersistentStoresAutomaticallyOption`

Key to automatically attempt to migrate versioned stores.

The corresponding value is an `NSNumber` object. If the `boolValue` of the number is `YES` and if the version hash information for the added store is determined to be incompatible with the model for the coordinator, Core Data will attempt to locate the source and mapping models in the application bundles, and perform a migration.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

`NSInferMappingModelAutomaticallyOption`

Key to attempt to create the mapping model automatically.

The corresponding value is an `NSNumber` object. If the `boolValue` of the number is `YES` and the value of the `NSMigratePersistentStoresAutomaticallyOption` is `YES`, the coordinator will attempt to infer a mapping model if none can be found.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Versioning Support

Keys in store metadata to support versioning.

```
NSString * const NSSStoreModelVersionHashesKey;
NSString * const NSSStoreModelVersionIdentifiersKey;
NSString * const NSPersistentStoreOSCompatibility;
```

Constants

`NSSStoreModelVersionHashesKey`

Key to represent the version hash information for the model used to create the store.

This key is used in the metadata for a persistent store.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

NSStoreModelVersionIdentifiersKey

Key to represent the version identifiers for the model used to create the store.

If you add your own annotations to a model's version identifier (see `versionIdentifiers`), they are stored in the persistent store's metadata. You can use this key to retrieve the identifiers from the metadata dictionaries available from `NSPersistentStore` (`metadata`) and `NSPersistentStoreCoordinator` (`metadataForPersistentStore:` (page 12) and related methods). The corresponding value is a Foundation collection (an `NSArray` or `NSSet` object).

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

NSPersistentStoreOSCompatibility

Key to represent the earliest version of Mac OS X the persistent store supports.

The corresponding value is an `NSNumber` object that takes the form of the constants defined by the Mac OS X availability macros (defined in `/usr/include/AvailabilityMacros.h`), for example 1040 represents Mac OS X version 10.4.0.

Backward compatibility may preclude some features.

Available in iOS 3.0 and later.

Declared in `NSPersistentStoreCoordinator.h`.

Notifications

NSPersistentStoreCoordinatorStoresDidChangeNotification

Posted whenever persistent stores are added to or removed from a persistent store coordinator, or when store UUIDs change.

The notification's object is the persistent store coordinator that was affected. The notification's *userInfo* dictionary contains information about the stores that were added or removed, specified using the following keys:

NSAddedPersistentStoresKey (page 18)	Key for the array of stores that were added.
NSRemovedPersistentStoresKey (page 18)	Key for the array of stores that were removed.
NSUUIDChangedPersistentStoresKey (page 18)	Key for the array of stores whose UUIDs changed.

Availability

Available in iOS 3.0 and later.

Declared In

`NSPersistentStoreCoordinator.h`

NSPersistentStoreCoordinatorWillRemoveStoreNotification

Posted whenever a persistent store is removed from a persistent store coordinator.

The notification is sent during the invocation of `NSPersistentStore`'s `willRemoveFromPersistentStoreCoordinator` method during store deallocation or removal. The notification's object is the persistent store coordinator will be removed.

Availability

Available in iOS 3.0 and later.

Declared In

NSPersistentStoreCoordinator.h

Document Revision History

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

Date	Notes
2009-05-01	Corrected typographical errors.
2009-02-12	Updated for Mac OS X v10.6.
2009-02-03	Updated for iOS 3.0.
2008-03-11	Corrected minor typographical errors.
2007-04-09	Updated for Mac OS X v10.5.
2006-05-23	First publication of this content as a separate document.

REVISION HISTORY

Document Revision History