Core Data Snippets

Data Management

2009-07-06

Contents

Introduction

This document contains prototypes for commonly-used snippets of code that you're likely to use in a program that uses Core Data. In some cases (particularly in cases where a code snippet might be only one or two lines), the document provides guidance rather than explicit code.

This document does not provide an in-depth explanation of the code snippets. You're expected to be familiar with the Core Data framework and understand how to use the snippets in context. To learn more about Core Data, read *Core Data Overview*.

Organization of This Document

The document contains the following articles:

- "Accessing the Core Data Stack" (page 7)
- "Fetching Managed Objects" (page 11)
- "Fetching Specific Property Values" (page 15)
- "Creating and Deleting Managed Objects" (page 19)

Accessing the Core Data Stack

This article contains snippets for creating and accessing parts of the main pieces of infrastructure defined by the Core Data framework.

Core Data Stack Architectures

How you access the parts of the Core Data stack may depend in part on the application architecture and platform.

Mac OS X Desktop

Broadly-speaking, on Mac OS X desktop there are two basic application architectures for programs that use Core Data:

- Single-coordinator applications.

  These applications typically have a single Core Data stack (as defined by a single persistent store coordinator) managed by a single controller object. They generally use a single persistent store for the whole application.

- Document-based applications.

  These applications typically use the Application Kit's `NSPersistentDocument` class. There is a usually a persistent store coordinator and a single persistent store associated with each document.

This article uses the terms "single-coordinator application" and "document-based application" to differentiate between these architectures.

iPhone

On iPhone, the application delegate usually maintains a persistent store coordinator that manages the application's store. It typically creates a managed object context, but it often doesn't own it. This is explained in greater detail in "Getting a Managed Object Context" (page 7).

Getting a Managed Object Context

On Mac OS X desktop:

- In an single-coordinator applications, you can get the application's context directly from the application delegate.

- In document-based applications, you can get the context directly from the document instance.

On iPhone:

- *By convention*, you can often get a context from a view controller. It's up to you, though, to follow this pattern.

  When you implement a view controller that integrates with Core Data, you can add an `NSManagedObjectContext` property.

  A view controller typically shouldn't retrieve the context from a global object such as the application delegate. This tends to make the application architecture rigid. Neither should a view controller typically create a context for its own use. This may mean that operations performed using the controller's context aren't registered with other contexts, so different view controllers will have different perspectives on the data.

  When you create a view controller, you pass it a context. You pass an existing context, or (in a situation where you want the new controller to manage a discrete set of edits) a new context that you create for it. It's typically the responsibility of the application delegate to create a context to pass to the first view controller that's displayed.

Sometimes, though, it's easier or more appropriate to retrieve the context from somewhere other than application or the document, or the view controller. Several objects you might use in a Core Data-based application keep a reference to a managed object context. A managed object itself has a reference to its own context, as do the various controller objects that support Core Data such as array and object controllers (`NSArrayController` and `NSObjectController` on Mac OS X desktop, and `NSFetchedResultsController` on iPhone).

Retrieving the context from one of these objects has the advantage that if you re-architect your application, for example to make use of multiple contexts, your code is likely to remain valid. For example, if you have a managed object, and you want to create a new managed object that will be related to it, you can ask original object for its managed object context and create the new object using that. This will ensure that the new object you create is in the same context as the original.

Creating a New Managed Object Context

You sometimes need to create a new managed object context to contain a disjoint set of edits that you might want to discard without perturbing the main context (for example, if you're presenting a modal view to add and edit a new object).

To create a new managed object context, you need a persistent store coordinator.

```
NSPersistentStoreCoordinator *psc = <#Get the coordinator#>;
NSManagedObjectContext *newContext = [[NSManagedObjectContext alloc] init];
[newContext setPersistentStoreCoordinator:psc];
```

If you already have a reference to an existing context, you can ask it for its persistent store coordinator. This way you can be sure that the new context is using the same coordinator as the existing one (assuming this is your intent):

```
NSManagedObjectContext *context = <#Get the context#>;
NSPersistentStoreCoordinator *psc = [context persistentStoreCoordinator];
NSManagedObjectContext *newContext = [[NSManagedObjectContext alloc] init];
```

```
[newContext setPersistentStoreCoordinator:psc];
```

Getting the Managed Object Model and Entities

You sometimes need to access a managed object model to get information about a particular entity.

Applications typically have just a single model (although it may have more than one configuration). In a single-coordinator application, you typically get the model directly from the application delegate. In a document-based application, you get the model directly from the document.

If you have access to a managed object context—directly or indirectly (see "Getting a Managed Object Context" (page 7))—you can get the model from the context's persistent store coordinator. From the model, you can retrieve an entity using `entitiesByName`:

```
NSManagedObjectContext *context = <#Get the context#>;
NSPersistentStoreCoordinator *psc = [context persistentStoreCoordinator];
NSManagedObjectModel *model = [psc managedObjectModel];
NSEntityDescription *entity = [[model entitiesByName] objectForKey:@"<#Entity
name#>"];
```

Creating a managed object: When you create a new managed object, you need to specify its entity. Typically, however, you don't actually need to access the entity or model directly—see "Creating and Deleting Managed Objects" (page 19).

Adding a Persistent Store

In many applications, there is only one persistent store for each persistent store coordinator. In an single-coordinator application, the store is associated with the whole application. In a document-based application, each document has a separate store. Sometimes, however, you might want to add another store. You add the store to the persistent store coordinator. You have to specify the store's type, location, and configuration (based on configurations present on the managed object model associated with the coordinator). You can also specify other options, such as whether an old version of the store should be migrated to a current version (see *Core Data Model Versioning and Data Migration Programming Guide*).

In an single-coordinator applications, you can get the application's coordinator directly from the application delegate.

In document-based applications, you can get the coordinator from the document's managed object context.

```
NSPersistentStoreCoordinator *psc = <#Get the coordinator#>;
NSURL *storeUrl = [NSURL fileURLWithPath:@"<#Path to store#>"];
NSString *storeType = <#Store type#>; // A store type, such as NSSQLiteStoreType
NSError *error = nil;
if (![psc addPersistentStoreWithType:storeType configuration:nil
    URL:storeUrl options:nil error:&error]) {
    // Handle the error
}
```

Fetching Managed Objects

This article contains snippets for fetching managed objects.

To fetch managed objects, you minimally need a managed object context against which to execute the fetch, and an entity description to specify the entity you want. You create an instance of `NSFetchRequest` and specify its entity. You may optionally specify an array of sort orderings and/or a predicate.

How you get the managed object context depends on your application architecture—see "Getting a Managed Object Context" (page 7). Once you have the context, though, you can get the entity using `NSEntityDescription`'s convenience method, `entityForName:inManagedObjectContext:`.

Basic Fetch

To get all the managed objects of a given entity, create a fetch request and specify just the entity:

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>"
    inManagedObjectContext:context];
[fetchRequest setEntity:entity];

NSError *error = nil;
NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest error:&error];
if (fetchedObjects == nil) {
    // Handle the error
}

[fetchRequest release];
```

Fetch with Sorting

To fetch managed objects in a particular order, in addition to the components in the basic fetch (described in "Basic Fetch" (page 11)) you need to specify an array of sort orderings:

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>"
    inManagedObjectContext:context];
[fetchRequest setEntity:entity];
```

```
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"<#Sort
 key#>"
    ascending:YES];
NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor,
nil];
[fetchRequest setSortDescriptors:sortDescriptors];

NSError *error = nil;
NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest error:&error];
if (fetchedObjects == nil) {
    // Handle the error
}

[fetchRequest release];
[sortDescriptor release];
[sortDescriptors release];
```

Fetch with a Predicate

To fetch managed objects that meet given criteria, in addition to the components in the basic fetch (described in "Basic Fetch" (page 11)) you need to specify a predicate:

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>"
    inManagedObjectContext:context];
[fetchRequest setEntity:entity];

NSPredicate *predicate = [NSPredicate predicateWithFormat:@"<#Predicate string#>",
    <#Predicate arguments#>];
[request setPredicate:predicate];

NSError *error = nil;
NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest error:&error];
if (fetchedObjects == nil) {
    // Handle the error
}

[fetchRequest release];
```

For more about predicates, see *Predicate Programming Guide*. For an alternative technique for creating the predicate that may be more efficient, see "Fetch with a Predicate Template" (page 12).

Fetch with a Predicate Template

To fetch managed objects that meet given criteria, in addition to the components in the basic fetch (described in "Basic Fetch" (page 11)) you need to specify a predicate. `NSPredicate`'s `predicateWithFormat:` method is typically the easiest way to use a predicate (as shown in "Fetch with a Predicate" (page 12)), but it's not the most efficient way to create the predicate itself. The predicate format string has to be parsed, arguments substituted, and so on. For performance-critical code, particularly if a given predicate is used repeatedly, you should consider other ways to create the predicate. For a predicate that you might use frequently, the easiest first step is to create a predicate template. You might create an accessor method that creates the predicate template lazily on demand:

```
// Assume an instance variable:
// NSPredicate *predicateTemplate;

- (NSPredicate *)predicateTemplate {
    if (predicateTemplate == nil) {
        predicateTemplate = [[NSPredicate predicateWithFormat:
    @"<#Key#> <#Operator#> <#$Variable#>"] retain];
    }
    return predicateTemplate;
}
```

When you need to use the template, you create a dictionary containing the substitution variables and generate the predicate using `predicateWithSubstitutionVariables:`.

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>"
    inManagedObjectContext:context];
[fetchRequest setEntity:entity];

NSDictionary *variables = [[NSDictionary alloc] initWithObjectsAndKeys:
    <#Value#>, @"<#Variable#>", nil];
NSPredicate *predicate = [[self predicateTemplate]
    predicateWithSubstitutionVariables:variables];
[request setPredicate:predicate];

NSError *error = nil;
NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest error:&error];
if (fetchedObjects == nil) {
    // Handle the error
}

[fetchRequest release];
[variables release];
```

For more about predicates, see *Predicate Programming Guide*.

Fetch with Sorting and a Predicate

To fetch managed objects that meet given criteria and in a particular order, in addition to the components in the basic fetch (described in "Basic Fetch" (page 11)) you need to specify a predicate and an array of sort orderings.

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
```

```
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>" inManagedObjectContext:context];
[fetchRequest setEntity:entity];

NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"<#Sort
 key#>" ascending:YES];
NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor,
nil];
[fetchRequest setSortDescriptors:sortDescriptors];

NSPredicate *predicate = [NSPredicate predicateWithFormat:@"<#Predicate string#>",
    <#Predicate arguments#>];
[request setPredicate:predicate];

NSError *error = nil;
NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest error:&error];
if (fetchedObjects == nil) {
    // Handle the error
}

[fetchRequest release];
[sortDescriptor release];
[sortDescriptors release];
```

For more about predicates, see *Predicate Programming Guide*. For an alternative technique for creating the predicate that may be more efficient, see "Fetch with a Predicate Template" (page 12).

Fetching Specific Property Values

This article contains snippets for fetching specific attribute values for a given entity.

Sometimes you don't want to fetch actual managed objects; instead, you just want to retrieve—for example—the largest or smallest value of a particular attribute, or distinct values for a given attribute. On iOS, you can use `NSExpressionDescription` objects to specify a function for a fetch request, and `setReturnsDistinctResults:` to return unique values.

To perform the fetch, you minimally need a managed object context against which to execute the fetch, and an entity description to specify the entity you want. How you get the managed object context depends on your application architecture—see "Getting a Managed Object Context" (page 7). Once you have the context, you can get the entity using `NSEntityDescription`'s convenience method, `entityForName:inManagedObjectContext:`.

Fetching Distinct Values

To fetch the unique values of a particular attribute across all instances of a given entity, you configure a fetch request with the method `setReturnsDistinctResults:` (and pass `YES` as the parameter). You also specify that the fetch should return dictionaries rather than managed objects, and the name of the property you want to fetch.

```
NSManagedObjectContext *context = <#Get the context#>;

NSEntityDescription *entity = [NSEntityDescription  entityForName:@"<#Entity
name#>" inManagedObjectContext:context];

NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity:entity];
[request setResultType:NSDictionaryResultType];
[request setReturnsDistinctResults:YES];
[request setPropertiesToFetch :[NSArray arrayWithObject:@"<#Attribute name#>"]];

// Execute the fetch.
NSError *error;
id requestedValue = nil;
NSArray *objects = [managedObjectContext executeFetchRequest:request
error:&error];
if (objects == nil) {
    // Handle the error.
}
```

Fetching Attribute Values that Satisfy a Given Function

To fetch values that satisfy a particular function (such as the maximum or minimum value), you use an instance of `NSExpressionDescription` to specify the property or properties you want to retrieve.

```
NSManagedObjectContext *context = <#Get the context#>;

NSFetchRequest *request = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"<#Entity
name#>" inManagedObjectContext:context];
[request setEntity:entity];

// Specify that the request should return dictionaries.
[request setResultType:NSDictionaryResultType];

// Create an expression for the key path.
NSExpression *keyPathExpression = [NSExpression expressionForKeyPath:@"<#Key-path
 for the property#>"];

// Create an expression to represent the function you want to apply
NSExpression *expression = [NSExpression expressionForFunction:@"<#Function
name#>"
    arguments:[NSArray arrayWithObject:keyPathExpression]];

// Create an expression description using the minExpression and returning a
date.
NSExpressionDescription *expressionDescription = [[NSExpressionDescription alloc]
 init];

// The name is the key that will be used in the dictionary for the return value.
[expressionDescription setName:@"<#Dictionary key#>"];
[expressionDescription setExpression:expression];
[expressionDescription setExpressionResultType:<#Result type#>]; // For example,
 NSDateAttributeType

// Set the request's properties to fetch just the property represented by the
expressions.
[request setPropertiesToFetch:[NSArray arrayWithObject:expressionDescription]];

// Execute the fetch.
NSError *error;
id requestedValue = nil;
NSArray *objects = [managedObjectContext executeFetchRequest:request
error:&error];
if (objects == nil) {
    // Handle the error.
}
else {
    if ([objects count] > 0) {
        requestedValue = [[objects objectAtIndex:0] valueForKey:@"<#Dictionary
 key#>"];
    }
}

[expressionDescription release];
[request release];
```

For a full list of supported functions, see `expressionForFunction:arguments:`.

If you want to retrieve multiple values simultaneously, create multiple instances of `NSExpressionDescription` to represent the different values you want to retrieve, and add them all to the array you pass in `setPropertiesToFetch:`. They must all, of course, apply to the same entity.

Creating and Deleting Managed Objects

This article contains snippets you use when creating or deleting a managed object.

Creating a Managed Object

When you create a new managed object, you need to specify its entity. Typically, however, you don't actually need access to the model directly. Instead, you can `NSEntityDescription`'s class method `insertNewObjectForEntityForName:inManagedObjectContext:` and pass the managed object context in which you want to create the new managed object. The method returns an instance of whatever class is defined in the managed object model to represent the entity, initialized with the default values given for its entity in the model.

To learn how to retrieve the managed object context, read "Getting a Managed Object Context" (page 7).

```
NSManagedObjectContext *context = <#Get the context#>;
<#Managed Object Class#> *newObject = [NSEntityDescription
    insertNewObjectForEntityForName:@"<#Entity name#>"
    inManagedObjectContext:context];
```

It is typically important to cast the new instance to the managed object class so that you can use the appropriate accessor methods without the compiler generating a warning (or, if you're using dot syntax, an error).

Saving a Managed Object

Simply creating a managed object does not cause it to be saved to a persistent store. It is simply associated with the managed object context. To commit changes to the store, you send the context a `save:` message.

To learn how to retrieve the managed object context, read "Getting a Managed Object Context" (page 7).

```
NSManagedObjectContext *context = <#Get the context#>;
NSError *error = nil;
if (![context save:&error]) {
    // Handle the error
}
```

Deleting a Managed Object

Simply being deallocated does not cause a managed object to be deleted from the persistent store. To delete a managed object you have to delete it from the context then save the context.

To learn how to retrieve the managed object context, read "Getting a Managed Object Context" (page 7)—or you can simply ask the object itself what context it belongs to.

```
NSManagedObject *aManagedObject = <#Get the managed object#>;
NSManagedObjectContext *context = [aManagedObject managedObjectContext];
[context deleteObject:aManagedObject];
NSError *error = nil;
if (![context save:&error]) {
    // Handle the error
}
```

Document Revision History

This table describes the changes to *Core Data Snippets*.

Date	Notes
2009-07-06	Updated for Mac OS X v10.6.
2009-05-03	Added discussion of how to use NSExpressionDescription.
2009-02-28	First iPhone version of a document that provides snippets of code that you can use when writing a program that uses Core Data.