Core Data Utility Tutorial

Cocoa > Data Management

2009-03-04

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

Simultaneously published in the United States and Canada.

Contents

Tables and Listings

Introduction to Core Data Utility Tutorial

This tutorial takes you through the steps of building a very basic Core Data–based command line utility. The goal is to illustrate the creation of a Core Data application entirely in code.

The task goal of this tutorial is to create a low-level Core Data-based utility. It simply records the date on which the utility is run, and its process ID, and prints the run history to the output. It shows the creation of a Core Data application entirely in code, including all aspects of the Core Data stack, instantiation of managed objects, and fetching—all without the distraction of a user interface—it even shows the creation of a model in code.

Who Should Read This Document

You will find this tutorial useful if you are using the Core Data framework to create a utility that does not have a user interface or if you want to gain a deeper understanding of the Core Data infrastructure.

Important: This tutorial is not intended for novice Cocoa developers. You must already be familiar with basic Cocoa development tools and techniques. This document does not repeat fundamental Cocoa programming concepts, nor does it provide explicit instructions for common operations in Xcode.

You should already be familiar with the fundamental Core Data architecture as described in Core Data Basics in *Core Data Programming Guide*. This tutorial is intended to reinforce the ideas explained in that article.

Organization of This Document

"Overview of the Tutorial" (page 9) describes the utility you will create, and the task constraints.

"Creating the Project" (page 11) describes how you create a Foundation Tool project in Xcode, and how you link in Core Data.

"Creating the Managed Object Model" (page 13) describes how you create the data model for the utility in code.

"The Application Log Directory" (page 19) illustrates one way to identify and if necessary create a directory in which to save the file for the utility's persistent store.

"Creating the Core Data Stack" (page 21) describes how to create and configure the managed object context and the persistent store coordinator in code.

"The Custom Managed Object Class" (page 25) specifies the Run entity and describes how to implement a custom managed object class.

"Listing Previous Runs" (page 31) describes how to fetch Run instances from the persistent store.

"Complete Source Listings" (page 33) shows the complete source code for the project.

See Also

You may want to read these documents if you want a better understanding of Core Data or if you want to use Core Data in a document–based application.

Core Data Programming Guide describes functionality provided by the Core Data framework from a high-level overview to in-depth descriptions.

NSPersistentDocument Core Data Tutorial takes you through the steps of building a document–based application that uses Core Data and Cocoa bindings.

Overview of the Tutorial

This tutorial takes you through the steps of building a very basic Core Data–based command-line utility. The goal is to illustrate the creation of a Core Data application entirely in code, including all aspects of the Core Data stack, instantiation of managed objects, and fetching—all without the distraction of a user interface. It even shows the creation of a model in code.

Task Goal

The task goal of this tutorial is to create a low-level Core Data–based utility. It simply records the date on which the utility is run, records its process ID, and prints the run history to the output. The utility uses a single entity, Run. The Run entity is very simple; it has only two attributes, the process ID and the date on which the process was run.

Note that the emphasis in this tutorial is on illustrating low-level functionality in Core Data, not on compactness, maintainability, or user friendliness. Although some explanation is given of what happens behind the scenes, it does not give an in-depth analysis of the Core Data infrastructure.

Creating the Project

This part of the tutorial guides you through creating the CDCLI project.

Create a New Project

Core Data is integrated into the Cocoa framework, so any Cocoa or Foundation application can use it. The CDCLI program you'll build is Foundation Tool that uses Core Data and garbage collection.

Create the project

Follow these steps to create the initial project:

1. Choose New Project from the File menu.
2. Select Foundation Tool in Xcode's project Assistant window (from the Command Line Utility section), and click the Next button.
3. Enter the project name (for example, "CDCLI") and a destination folder for the project.

Note: The source file created by the Assistant that contains the `main` function is hereafter referred to as "the main source file."

Link the Core Data framework

1. Link the Core Data framework to the project.

   In Xcode, you can double-click the CDCLI Target to open the Target Info pane. In the General tab you can then add CoreData.framework to the list of Linked Libraries. See Files in Projects in *Xcode Project Management Guide* for details.

2. Add an import statement to the main source file (`#import <CoreData/CoreData.h>`).

Adopt Garbage Collection

Remove the references to autorelease pools; add a function call to start the garbage collector; and change the project build settings to use garbage collection:

1. In the main source file, *remove* the lines that create and drain the autorelease pool:

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
[pool drain];
```

2. At the beginning of the main source file, add a function call to start the garbage collector:

```
objc_startCollectorThread();
```

3. In the project's Build settings, change the garbage collection setting to Required (see "Enabling Garbage Collection" in *Garbage Collection Programming Guide*).

What Happened?

You created a very simple Foundation Tool project and added the Core Data framework. There is no need to use the Application Kit when using Core Data. It is not even necessary to use the Xcode data modeling tool. In the next chapter you will create the model entirely in code. Using the modeling tool does, however, typically save you a lot of time and effort.

Creating the Managed Object Model

This chapter specifies the Run entity and shows you how to create the managed object model. Although it is typically easiest to create the model in Xcode, in this tutorial you create the model entirely in code.

Xcode has a data modeling tool that you typically use to define the schema for your application (see *Xcode Tools for Core Data* for full details). The Xcode data modeling tool is analogous to Interface Builder in that it allows you to graphically create a complex collection of objects that are archived and at runtime are unarchived. Creating a user interface without Interface Builder is possible, but can require a lot of effort. Similarly, even a reasonably straightforward model requires a lot of code, so this tutorial only uses a single entity with two simple attributes. For more about creating a model using Xcode, see *Creating a Managed Object Model with Xcode*.

Specifying the Entity

The Run entity has two attributes, the process ID and the date on which the process was run. Neither attribute is optional—that is, each must have a value if an instance is to be considered valid (and if you try to save an instance without a value, you will get a validation error). The process ID has a default value of `-1`. In conjunction with the validation rules, this ensures that the value is properly set at runtime. You must also specify the class that will represent the entity in the utility—in this example you will use a custom class named "Run".

Table 3-1 Attributes for the Run entity

Name	Type	Optional	Default Value	Minimum Value
date	date	NO		
processID	int	NO	-1	0

Create the Managed Object Model

You could create the model in Xcode, put it in the application support directory, and load it at runtime using NSManagedObjectModel's `initWithContentsOfURL:`. This example, however, illustrates how to create the model entirely in code.

Create the Model Instance

The first step is to create the managed object model instance itself, if necessary.

1. At the top of the main source file, before `main` add a declaration for the function `NSManagedObjectModel *managedObjectModel()`.

2. In the main source file, implement the `managedObjectModel` function. It declares a static variable for the managed object model, and returns it immediately if it is not `nil`. If it is `nil`, create a new managed object model, then return it as the function result.

```
NSManagedObjectModel *managedObjectModel() {

    static NSManagedObjectModel *mom = nil;

    if (mom != nil) {
        return mom;
    }

    mom = [[NSManagedObjectModel alloc] init];
    // implementation continues...
    return mom;
}
```

You should enter the code described in the following sections, "Create the Entity" (page 14) and "Add the Attributes" (page 14), immediately before the return statement (where the comment states, "implementation continues...").

Create the Entity

The first step after creating the model itself, is to create the entity. You must set the name of the entity object before you add it to the model.

1. Create the entity description object, set its name and managed object class name, and add it to the model as follows:

```
NSEntityDescription *runEntity = [[NSEntityDescription alloc] init];
[runEntity setName:@"Run"];
[runEntity setManagedObjectClassName:@"Run"];
[mom setEntities:[NSArray arrayWithObject:runEntity]];
```

Add the Attributes

Attributes are represented by instances of `NSAttributeDescription`. You must create two instances—one for the date, the other for the process ID—and set their characteristics appropriately. Both require a name and a type, and neither is optional. The process ID has a default value of `-1`. You also need to create a predicate for the process ID validation.

1. Create the date attribute description object as follows. Its type is `NSDateAttributeType` and it is not optional.

```
NSAttributeDescription *dateAttribute = [[NSAttributeDescription alloc] init];

[dateAttribute setName:@"date"];
[dateAttribute setAttributeType:NSDateAttributeType];
[dateAttribute setOptional:NO];
```

2. Create the process ID attribute description object as follows. Its type is `NSInteger32AttributeType`, it is not optional, and its default value is `-1`.

```
NSAttributeDescription *idAttribute = [[NSAttributeDescription alloc] init];

[idAttribute setName:@"processID"];
[idAttribute setAttributeType:NSInteger32AttributeType];
[idAttribute setOptional:NO];
[idAttribute setDefaultValue:[NSNumber numberWithInteger:-1]];
```

3. Create the validation predicate for the process ID. The value of the attribute itself must be greater than zero. The following code is equivalent to `validationPredicate = [NSPredicate predicateWithFormat:@"SELF > 0"]`, but this example continues the theme of illustrating the long-hand form.

```
NSExpression *lhs = [NSExpression expressionForEvaluatedObject];
NSExpression *rhs = [NSExpression expressionForConstantValue:
                                        [NSNumber numberWithInteger:0]];

NSPredicate *validationPredicate = [NSComparisonPredicate
                                       predicateWithLeftExpression:lhs
                                       rightExpression:rhs
                                       modifier:NSDirectPredicateModifier
                                      type:NSGreaterThanPredicateOperatorType
                                       options:0];
```

4. Each validation predicate requires a corresponding error string. Typically the error string should be appropriately localized. You can either provide a localized representation here (using, for example, `NSLocalizedString`) or supply a localization dictionary for the model. The latter is shown in the next section ("Add a Localization Dictionary" (page 15)). You provide the attribute description with an array of predicates and an array of error strings. In this case, each array contains just a single object.

```
NSString *validationWarning = @"Process ID < 1";
[idAttribute setValidationPredicates:[NSArray arrayWithObject:validationPredicate]
        withValidationWarnings:[NSArray arrayWithObject:validationWarning]];
```

5. Finally, set the properties for the entity.

```
NSArray *properties = [NSArray arrayWithObjects: dateAttribute, idAttribute,
nil];
[runEntity setProperties:properties];
```

Add a Localization Dictionary

You can set a localization dictionary to provide localized string values for entities, properties, and error strings related to the model. The key and value pattern is described in the API reference for `setLocalizationDictionary:`. The string you use as the key for the error must be the same as that you specified for the corresponding validation predicate.

```
NSMutableDictionary *localizationDictionary = [NSMutableDictionary dictionary];

[localizationDictionary setObject:@"Date" forKey:@"Property/date/Entity/Run"];
[localizationDictionary setObject:@"Process ID"
forKey:@"Property/processID/Entity/Run"];
```

```
[localizationDictionary setObject:@"Process ID must not be less than 1"
forKey:@"ErrorString/Process ID < 1"];

[mom setLocalizationDictionary:localizationDictionary];
```

Instantiate a Managed Object Model

So that you can test the implementation thus far, instantiate the managed object model and log its description of the model.

1. In the `main` function, after the garbage collector is started, declare a variable of class `NSManagedObjectModel` and assign its value to the result of invoking the `managedObjectModel` function. Print the model description using `NSLog`.

   ```
   NSManagedObjectModel *mom = managedObjectModel();
   NSLog(@"The managed object model is defined as follows:\n%@", mom);
   ```

Build and Test

Build and run the utility. It should compile without warnings. The logged description of the model file should contain the entity and attributes you defined. Note that at this stage the model has not yet been used, so its `isEditable` state remains true.

Complete Listing

The complete listing of the `managedObjectModel` function is shown in Listing 3-1.

Listing 3-1 Complete listing of the `managedObjectModel` function

```
NSManagedObjectModel *managedObjectModel() {

    static NSManagedObjectModel *mom = nil;

    if (mom != nil) {
        return mom;
    }

    mom = [[NSManagedObjectModel alloc] init];

    NSEntityDescription *runEntity = [[NSEntityDescription alloc] init];
    [runEntity setName:@"Run"];
    [runEntity setManagedObjectClassName:@"Run"];
    [mom setEntities:[NSArray arrayWithObject:runEntity]];

    NSAttributeDescription *dateAttribute = [[NSAttributeDescription alloc]
init];
```

```
    [dateAttribute setName:@"date"];
    [dateAttribute setAttributeType:NSDateAttributeType];
    [dateAttribute setOptional:NO];


   NSAttributeDescription *idAttribute = [[NSAttributeDescription alloc] init];

    [idAttribute setName:@"processID"];
    [idAttribute setAttributeType:NSInteger32AttributeType];
    [idAttribute setOptional:NO];
    [idAttribute setDefaultValue:[NSNumber numberWithInteger:-1]];

    NSExpression *lhs = [NSExpression expressionForEvaluatedObject];
    NSExpression *rhs = [NSExpression expressionForConstantValue:
                                      [NSNumber numberWithInteger:0]];

    NSPredicate *validationPredicate = [NSComparisonPredicate
                                          predicateWithLeftExpression:lhs
                                          rightExpression:rhs
                                          modifier:NSDirectPredicateModifier

type:NSGreaterThanPredicateOperatorType
                                          options:0];

    NSString *validationWarning = @"Process ID < 1";

    [idAttribute setValidationPredicates:[NSArray
arrayWithObject:validationPredicate]
        withValidationWarnings:[NSArray arrayWithObject:validationWarning]];

   NSArray *properties = [NSArray arrayWithObjects: dateAttribute, idAttribute,
 nil];
    [runEntity setProperties:properties];

    NSMutableDictionary *localizationDictionary = [NSMutableDictionary
dictionary];

    [localizationDictionary setObject:@"Date"
                            forKey:@"Property/date/Entity/Run"];
    [localizationDictionary setObject:@"Process ID"
                            forKey:@"Property/processID/Entity/Run"];
    [localizationDictionary setObject:@"Process ID must not be less than 1"
                            forKey:@"ErrorString/Process ID < 1"];

    [mom setLocalizationDictionary:localizationDictionary];

    return mom;
}
```

The Application Log Directory

The utility needs somewhere to save the file for the persistent store. This section illustrates one way to identify and if necessary create an appropriate directory. Although it is a useful abstraction for the utility, this is not directly relevant to Core Data, so no additional explanation is given.

The applicationLogDirectory Function

This section illustrates a simple means to identify and if necessary create a directory (in `~/Library/Logs`—the Logs directory in your home directory) in which to save the file for the persistent store.

In the main source file, before `main` declare a function, `applicationLogDirectory()`, that returns an `NSString` object, then implement it as follows:

```
NSString *applicationLogDirectory() {
    NSString *LOG_DIRECTORY = @"CDCLI";
    static NSString *ald = nil;

    if (ald == nil) {
        NSArray *paths = NSSearchPathForDirectoriesInDomains
                (NSLibraryDirectory, NSUserDomainMask, YES);
        if ([paths count] == 1) {
            ald = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Logs"];
            ald = [ald stringByAppendingPathComponent:LOG_DIRECTORY];
            NSFileManager *fileManager = [NSFileManager defaultManager];
            BOOL isDirectory = NO;
            if (![fileManager fileExistsAtPath:ald isDirectory:&isDirectory])
{
                if (![fileManager createDirectoryAtPath:ald attributes:nil]) {
                    ald = nil;
                }
            }
            else {
                if (!isDirectory) {
                    ald = nil;
                }
            }
        }
    }
    return ald;
}
```

Update the main Function

In the `main` function, after the invocation of the `managedObjectModel` function, invoke `applicationLogDirectory()`, and ensure that it does not return `nil`. If it does, report an error and exit.

```
if (applicationLogDirectory() == nil) {
    NSLog(@"Could not find application log directory\nExiting...");
    exit(1);
}
```

Build and Test

Build and run the utility. It should compile without warnings. The application log directory should be created correctly, and no errors should be logged.

Creating the Core Data Stack

This chapter shows you how to create and configure the Core Data stack, from the managed object context to the underlying persistent store. Creating the context is easy—you simply have to allocate and initialize an instance of `NSManagedObjectContext`. The more complex part is the remainder of the configuration. You must create and configure a persistent store coordinator, and then set up the persistent stores.

The managed object context is responsible for managing the object graph. The task of managing the persistent stores falls to the persistent store coordinator. Its job is to mediate between the managed object context or contexts and the persistent store or stores. It presents a façade to the contexts, representing a collection of stores as a single virtual store. In this example, the coordinator manages just a single store.

To add a store, you use the `NSPersistentStoreCoordinator` method `addPersistentStoreWithType:configuration:URL:options:error:`. This returns an object representing the new store, or `nil` if it cannot be created. (There is currently no public API to manipulate store instances; they may be used, however, as arguments to other methods of `NSPersistentStoreCoordinator`.) You must specify both the store's location in the file system and its type (this example does not make use of model configurations). In this example it is an XML store—because its reasonably human-readable form facilitates testing. Note that the file name extension is not `.xml`. You should avoid using generic file extensions—consider what would happen if all applications used the same extension. . .

The managedObjectContext Function

The main purpose of the `managedObjectContext` function is to return a properly configured managed object context. In this example, in order to do so you must also configure the remainder of the Core Data stack.

Create the Context Instance

The first step is to create the managed object context instance itself, if necessary.

1. At the top of the main source file, before `main` add a declaration for the function `NSManagedObjectContext *managedObjectContext()`.

2. In the main source file, implement the `managedObjectContext` function. Declare a static variable for the context. If the variable is not `nil` return it immediately. If it is `nil`, create a new context, then return it as the function result.

```
NSManagedObjectContext *managedObjectContext()
{
    static NSManagedObjectContext *moc = nil;
    if (moc != nil) {
        return moc;
    }
```

```
    moc = [[NSManagedObjectContext alloc] init];

    // implementation continues...

    return moc;
}
```

Set up the Persistent Store Coordinator and Store

The second step is to create the persistent store coordinator and configure the persistent store.

1. Create a persistent store coordinator, then set the coordinator for the context.

```
NSPersistentStoreCoordinator *coordinator =
        [[NSPersistentStoreCoordinator alloc]
                initWithManagedObjectModel: managedObjectModel()];
[moc setPersistentStoreCoordinator: coordinator];
```

2. Create a new persistent store of the appropriate type. If for some reason the store cannot be created, log an appropriate warning.

```
NSString *STORE_TYPE = NSXMLStoreType;
NSString *STORE_FILENAME = @"CDCLI.cdcli";

NSError *error;
NSURL *url = [NSURL fileURLWithPath:
        [applicationLogDirectory()
stringByAppendingPathComponent:STORE_FILENAME]];

NSPersistentStore *newStore = [coordinator addPersistentStoreWithType:STORE_TYPE
                                        configuration:nil
                                                  URL:url
                                              options:nil
                                                error:&error];

if (newStore == nil) {
    NSLog(@"Store Configuration Failure\n%@",
            ([error localizedDescription] != nil) ?
            [error localizedDescription] : @"Unknown Error");
}
```

Instantiate a Managed Object Context

So that you can test the implementation thus far, instantiate the managed object context.

1. In the `main` function, after the line in which the description of the managed object model is logged, declare a variable of type `NSManagedObjectContext` and assign its value to the result of invoking the `managedObjectContext` function.

```
NSManagedObjectContext *moc = managedObjectContext();
```

Build and Test

Build and run the utility. It should compile without errors, although you should get a warning that the variable `moc` is unused in the `main` function. When you run the utility, the `managedObjectContext` function should not log any errors.

Complete Listing

The complete listing of the `managedObjectContext` function is shown in Listing 5-1.

Listing 5-1 Complete listing of the `managedObjectContext` function

```
NSManagedObjectContext *managedObjectContext()
{
    static NSManagedObjectContext *moc = nil;

    if (moc != nil) {
        return moc;
    }

    moc = [[NSManagedObjectContext alloc] init];

    NSPersistentStoreCoordinator *coordinator =
        [[NSPersistentStoreCoordinator alloc]
                initWithManagedObjectModel: managedObjectModel()];
    [moc setPersistentStoreCoordinator: coordinator];

    NSString *STORE_TYPE = NSXMLStoreType;
    NSString *STORE_FILENAME = @"CDCLI.cdcli";

    NSError *error;
    NSURL *url = [NSURL fileURLWithPath:
        [applicationLogDirectory()
stringByAppendingPathComponent:STORE_FILENAME]];

    NSPersistentStore *newStore = [coordinator
addPersistentStoreWithType:STORE_TYPE
                                            configuration:nil
                                                      URL:url
                                                  options:nil
                                                    error:&error];

    if (newStore == nil) {
        NSLog(@"Store Configuration Failure\n%@",
                ([error localizedDescription] != nil) ?
                [error localizedDescription] : @"Unknown Error");
    }
    return moc;
}
```

The Custom Managed Object Class

The managed object model for this tutorial specifies that the Run entity is represented by a custom class, Run. This chapter shows how to implement the class that uses a scalar value to represent one of its attributes, and how to define custom accessor methods and an initializer that is invoked only when a new instance is first created.

Typically there is no need to add instance variables—it is usually better to let the Core Data framework manage properties—for the purposes of illustration, however, in this example you will use a scalar value for the process ID attribute. You must implement custom accessor methods for any attributes you choose to represent using scalar values.

One drawback with using scalar instance variables is that there is no unambiguous way to represent a `nil` value. The `NSKeyValueCoding` protocol defines a special method—`setNilValueForKey:`—that allows you to specify what happens if an attempt is made to set a scalar value to `nil`.

There are a number of different situations in which you might want to initialize a managed object. You might want to perform initialization every time an instance of a given class is created, in which case you can simply override the designated initializer. You might also, though, want to perform different initialization whenever an object is retrieved from a persistent store or—perhaps more commonly—only when an object is first created. Core Data provides special methods to cater for both situations—`awakeFromFetch` and `awakeFromInsert` respectively. This example illustrates the latter case: You want to record the date and time when a new record is created and not update the value thereafter.

Implementing the Managed Object Subclass

Create the Class Files

The first step is to create the files for the new class. If you had a managed object model as a project resource, you could use the New File assistant to create a managed object class from an entity in the model. In this case, however, you do not, so create the files as you would for any other Objective-C class.

1. In Xcode, add a new Objective-C class file (.h and .m files) for the Run class.

2. In the `Run.h` file, set the class's superclass to `NSManagedObject`, and declare properties for `date`, `primitiveDate`, and `processID` (`primitiveDate` is used in `awakeFromInsert`—see "Implement the Initializer" (page 27)). Add an instance variable of type `NSInteger` for the process ID.

```
@interface Run : NSManagedObject {
    NSInteger processID;
}
@property (retain) NSDate *date;
@property (retain) NSDate *primitiveDate;
@property NSInteger processID;
@end
```

Implement the Accessor Methods

Core Data automatically implements accessors for managed object properties at runtime, so typically you don't have to implement them yourself. When you do, though, (such as for scalar attributes) you must invoke the appropriate access and change notification methods. In the implementation block in the `Run.m` file, do the following:

1. Core Data automatically implements accessors for the date attribute at runtime. To suppress compiler warnings, though, declare the date properties as dynamic.

```
@dynamic date, primitiveDate;
```

2. Implement a get accessor for the process ID. You retrieve the value from the managed object's instance variable. You invoke the appropriate access notification methods to ensure that if the receiver is a fault, the value is retrieved from the persistent store.

```
- (int)processID {
    [self willAccessValueForKey:@"processID"];
    NSInteger pid = processID;
    [self didAccessValueForKey:@"processID"];
    return pid;
}
```

3. Implement a set accessor for the process ID. You set the value of the managed object's instance variable. You must also invoke the appropriate change notification methods.

```
- (void)setProcessID:(int)newProcessID {
    [self willChangeValueForKey:@"processID"];
    processID = newProcessID;
    [self didChangeValueForKey:@"processID"];
}
```

Dealing With nil Values

If you represent an attribute using a scalar value, you need to specify what happens if the value is set to `nil` using key-value coding. You do this with the `setNilValueForKey:` method. In this case, simply set the process ID to `0`.

1. Implement a `setNilValueForKey:` method. If the key is "processID" then set `processID` to `0`.

```
- (void)setNilValueForKey:(NSString *)key {

    if ([key isEqualToString:@"processID"]) {
        self.processID = 0;
    }
    else {
        [super setNilValueForKey:key];
    }
}
```

Implement the Initializer

`NSManagedObject` provides a special method—`awakeFromInsert`—that is invoked only when a new managed object is first created (strictly, when it is inserted into the managed object context) and *not* when it is subsequently fetched from a persistent store. You can use it here to record the date and time when a new record is created (the value won't then be updated when an object is fetched).

1. Implement an `awakeFromInsert` method that sets the receiver's date to the current date and time.

```
- (void) awakeFromInsert {
    [super awakeFromInsert];
    self.primitiveDate = [NSDate date];
}
```

You use the primitive accessor in `awakeFromInsert` to change to the date. The primitive accessors do not emit KVO notifications that cause the change to be recorded as a separate undo event.

Create an Instance of the Run Entity

To create a new instance of a given entity and insert it into a managed object context, you usually use the `NSEntityDescription` convenience method `insertNewObjectForEntityForName:inManagedObjectContext:`. The advantage of using the convenience method is that it's convenient! In this case, though, you'll perform the set-up operations yourself. Given the new instance, you can set its process ID to the ID of the current process, then send the managed object context a save message to commit the change to the persistent store.

1. In the `main` source file, import the header for the Run class.

```
#import "Run.h"
```

2. In the main function, after the invocation of the `managedObjectContext()` function, create a new instance of the Run class. You must retrieve the Run entity description from the managed object model so that you can tell the new managed object of what entity it is an instance.

```
NSEntityDescription *runEntity = [[mom entitiesByName] objectForKey:@"Run"];
Run *run = [[Run alloc] initWithEntity:runEntity
insertIntoManagedObjectContext:moc];
```

3. Get the process ID of the current process, and set the process ID of the Run object.

```
NSProcessInfo *processInfo = [NSProcessInfo processInfo];
run.processID = [processInfo processIdentifier];
```

4. Commit the changes to the persistent store by saving the managed object context. Check for any errors, and exit if an error occurs.

```
NSError *error = nil;

if (![moc save: &error]) {
    NSLog(@"Error while saving\n%@",
        ([error localizedDescription] != nil) ? [error localizedDescription] :
 @"Unknown Error");
```

```
        exit(1);
    }
```

Build and Test

Build and run the utility. It should compile without warnings. When you run the utility, it should not log any errors. You should see a new file created in the application log directory. If you inspect the file, you should see that it contains details of run objects.

Test some of the other features. Comment out the line that sets the Run object's process ID. Build and run the utility. What happens (recall that the default value for the process ID is `-1`)? Do you see the localized error message (defined in "Add a Localization Dictionary" (page 15))? Use key-value coding to set the process ID to `nil`. Build and run the utility. Again, what happens? And finally, comment out the `setNilValueForKey:` method and test once more.

Complete Listings

The Run Class

A complete listing of the declaration and implementation of the Run class is shown in Listing 6-1.

Listing 6-1 Complete listing of the declaration and implementation of the Run class

```
@interface Run : NSManagedObject
{
    NSInteger processID;
}

@property (retain) NSDate *date;
@property (retain) NSDate *primitiveDate;
@property NSInteger processID;

@end


@implementation Run

@dynamic date, primitiveDate;

- (void) awakeFromInsert{
    // set date to now
    self.primitiveDate = [NSDate date];
}


- (int)processID {
    [self willAccessValueForKey:@"processID"];
    NSInteger pid = processID;
    [self didAccessValueForKey:@"processID"];
```

```
    return pid;
}


- (void)setProcessID:(int)newProcessID {
    [self willChangeValueForKey:@"processID"];
    processID = newProcessID;
    [self didChangeValueForKey:@"processID"];
}


- (void)setNilValueForKey:(NSString *)key {
    if ([key isEqualToString:@"processID"]) {
        self.processID = 0;
    }
    else {
        [super setNilValueForKey:key];
    }
}

@end
```

The main() Function

The main function is shown in Listing 6-2.

Listing 6-2 Listing of the `main` function

```
int main (int argc, const char * argv[]) {

    objc_startCollectorThread();

    NSManagedObjectModel *mom = managedObjectModel();
    NSLog(@"mom: %@", mom);

    if (applicationLogDirectory() == nil) {
        NSLog(@"Could not find application logs directory\nExiting...");
        exit(1);
    }

    NSManagedObjectContext *moc = managedObjectContext();

   NSEntityDescription *runEntity = [[mom entitiesByName] objectForKey:@"Run"];
    Run *run = [[Run alloc] initWithEntity:runEntity
insertIntoManagedObjectContext:moc];

    NSProcessInfo *processInfo = [NSProcessInfo processInfo];
    run.processID = [processInfo processIdentifier];

    NSError *error = nil;

    if (![moc save: &error]) {
        NSLog(@"Error while saving\n%@",
           ([error localizedDescription] != nil) ? [error localizedDescription]
 : @"Unknown Error");
        exit(1);
```

```
    }

    // Implementation will continue...

    return 0;
}
```

Listing Previous Runs

This section shows you how to fetch all the Run instances from the persistent store.

Fetching Run Objects

Create and Execute the Fetch Request

The first step is to create the fetch request. You want to fetch instances of the Run entity and order the results by recency. You need to set the entity for the fetch request to be the Run entity, and create and set an appropriate array of sort orderings. Finally, you perform the fetch by sending the managed object context an `executeFetchRequest:request error:` message.

1. In the `main` function, immediately after the code you added in the previous chapter, create a new fetch request and set the entity (recall that in the previous chapter you retrieved the Run entity description to create the new instance of Run).

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];
[request setEntity:runEntity];
```

2. Create a new sort descriptor to arrange the fetch results by recency. Set the sort descriptor for the fetch—note that you must supply an array of sort descriptors.

```
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
        initWithKey:@"date" ascending:YES];
[request setSortDescriptors:[NSArray arrayWithObject:sortDescriptor]];
```

3. Execute the fetch request by sending it to the managed object context. Recall that you declared an error in the previous chapter. If there is an error, report it and exit.

```
error = nil;
NSArray *array = [moc executeFetchRequest:request error:&error];
if ((error != nil) || (array == nil))
{
    NSLog(@"Error while fetching\n%@",
          ([error localizedDescription] != nil) ? [error localizedDescription]
 : @"Unknown Error");
    exit(1);
}
```

Display the Results

Iterate through the array of fetched run objects and log the run information.

1. Create a date formatter object to display the time information.

```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setDateStyle:NSDateFormatterMediumStyle];
[formatter setTimeStyle:NSDateFormatterMediumStyle];
```

2. Print out the run history for the process.

```
NSLog(@"%@ run history:", [processInfo processName]);

for (run in array)
{
    NSLog(@"On %@ as process ID %d",
            [formatter stringForObjectValue:run.date],
            run.processID);
}
```

Build and Test

Build and run the utility. It should compile without warnings. When you run the utility, it should not log any errors. It should properly display the run history.

Complete Source Listings

main

A complete listing of the `main` source file for the finished tutorial is shown in Listing 8-1.

Listing 8-1 Complete listing of the `main` source file

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

#import "Run.h"

NSManagedObjectModel *managedObjectModel();
NSString *applicationLogDirectory();
NSManagedObjectContext *managedObjectContext();


int main (int argc, const char * argv[])
{

    objc_startCollectorThread();

    NSManagedObjectModel *mom = managedObjectModel();
    NSLog(@"mom: %@", mom);

    if (applicationLogDirectory() == nil) {
        NSLog(@"Could not find application support directory\nExiting...");
        exit(1);
    }

    NSManagedObjectContext *moc = managedObjectContext();

   NSEntityDescription *runEntity = [[mom entitiesByName] objectForKey:@"Run"];
    Run *run = [[Run alloc] initWithEntity:runEntity
            insertIntoManagedObjectContext:moc];

    NSProcessInfo *processInfo = [NSProcessInfo processInfo];
    [run setProcessID:[processInfo processIdentifier]];

    NSError *error = nil;
    if (![managedObjectContext() save: &error]) {
        NSLog(@"Error while saving\n%@",
           ([error localizedDescription] != nil) ? [error localizedDescription]
 : @"Unknown Error");
        exit(1);
    }


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

```
    [request setEntity:runEntity];

    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
            initWithKey:@"date" ascending:YES];

    [request setSortDescriptors:[NSArray arrayWithObject:sortDescriptor]];

    error = nil;
    NSArray *array = [moc executeFetchRequest:request error:&error];

    if ((error != nil) || (array == nil)) {
        NSLog(@"Error while fetching\n%@",
                ([error localizedDescription] != nil)
                ? [error localizedDescription] : @"Unknown Error");
        exit(1);
    }


    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    [formatter setDateStyle:NSDateFormatterMediumStyle];
    [formatter setTimeStyle:NSDateFormatterMediumStyle];

    NSLog(@"%@ run history:", [processInfo processName]);
    for (run in array) {
        NSLog(@"On %@ as process ID %d",
                [formatter stringForObjectValue:run.date],
                run.processID);
    }

    return 0;
}



NSString *applicationLogDirectory() {
    NSString *LOG_DIRECTORY = @"CDCLI";
    static NSString *ald = nil;

    if (ald == nil) {
        NSArray *paths = NSSearchPathForDirectoriesInDomains
                (NSLibraryDirectory, NSUserDomainMask, YES);

        if ([paths count] == 1) {
            ald = [[paths objectAtIndex:0]
stringByAppendingPathComponent:@"Logs"];
            ald = [ald stringByAppendingPathComponent:LOG_DIRECTORY];
            NSFileManager *fileManager = [NSFileManager defaultManager];
            BOOL isDirectory = NO;

            if (![fileManager fileExistsAtPath:ald isDirectory:&isDirectory])
{
                if (![fileManager createDirectoryAtPath:ald attributes:nil]) {
                    ald = nil;
                }
            }
            else {
                if (!isDirectory) {
                    ald = nil;
```

```
                }
            }
        }
    }
    return ald;
}


NSManagedObjectModel *managedObjectModel() {

    static NSManagedObjectModel *mom = nil;

    if (mom != nil) {
        return mom;
    }

    mom = [[NSManagedObjectModel alloc] init];

    NSEntityDescription *runEntity = [[NSEntityDescription alloc] init];
    [runEntity setName:@"Run"];
    [runEntity setManagedObjectClassName:@"Run"];
    [mom setEntities:[NSArray arrayWithObject:runEntity]];

    NSAttributeDescription *dateAttribute;

    dateAttribute = [[NSAttributeDescription alloc] init];

    [dateAttribute setName:@"date"];
    [dateAttribute setAttributeType:NSDateAttributeType];
    [dateAttribute setOptional:NO];


    NSAttributeDescription *idAttribute;

    idAttribute = [[NSAttributeDescription alloc] init];

    [idAttribute setName:@"processID"];
    [idAttribute setAttributeType:NSInteger32AttributeType];
    [idAttribute setOptional:NO];
    [idAttribute setDefaultValue:[NSNumber numberWithInteger:-1]];

    NSExpression *lhs = [NSExpression expressionForEvaluatedObject];
    NSExpression *rhs = [NSExpression expressionForConstantValue:
                                        [NSNumber numberWithInteger:0]];

    NSPredicate *validationPredicate = [NSComparisonPredicate
                                            predicateWithLeftExpression:lhs
                                            rightExpression:rhs
                                            modifier:NSDirectPredicateModifier

type:NSGreaterThanPredicateOperatorType
                                            options:0];

    NSString *validationWarning = @"Process ID < 1";

    [idAttribute setValidationPredicates:[NSArray
arrayWithObject:validationPredicate]
```

```
        withValidationWarnings:[NSArray arrayWithObject:validationWarning]];

    [runEntity setProperties:
            [NSArray arrayWithObjects: dateAttribute, idAttribute, nil]];

    NSMutableDictionary *localizationDictionary = [NSMutableDictionary
dictionary];

    [localizationDictionary setObject:@"Date"
                           forKey:@"Property/date/Entity/Run"];
    [localizationDictionary setObject:@"Process ID"
                           forKey:@"Property/processID/Entity/Run"];
    [localizationDictionary setObject:@"Process ID must not be less than 1"
                           forKey:@"ErrorString/Process ID < 1"];

    [mom setLocalizationDictionary:localizationDictionary];

    return mom;
}



NSManagedObjectContext *managedObjectContext() {

    static NSManagedObjectContext *moc = nil;

    if (moc != nil) {
        return moc;
    }

    moc = [[NSManagedObjectContext alloc] init];

    NSPersistentStoreCoordinator *coordinator =
        [[NSPersistentStoreCoordinator alloc]
            initWithManagedObjectModel: managedObjectModel()];
    [moc setPersistentStoreCoordinator: coordinator];

    NSString *STORE_TYPE = NSXMLStoreType;
    NSString *STORE_FILENAME = @"CDCLI.cdcli";

    NSError *error;
    NSURL *url = [NSURL fileURLWithPath:
        [applicationLogDirectory()
stringByAppendingPathComponent:STORE_FILENAME]];

    NSPersistentStore *newStore = [coordinator
addPersistentStoreWithType:STORE_TYPE
                                            configuration:nil
                                                      URL:url
                                                  options:nil
                                                    error:&error];

    if (newStore == nil) {
        NSLog(@"Store Configuration Failure\n%@",
                ([error localizedDescription] != nil) ?
                [error localizedDescription] : @"Unknown Error");
    }
    return moc;
```

```
}
```

The Run Class

Complete listings of the declaration and implementation of the Run class for the finished tutorial are shown in Listing 8-2 and Listing 8-3 respectively.

Listing 8-2 Listing of the declaration of the Run class

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@interface Run : NSManagedObject {
    NSInteger processID;
}

@property (retain) NSDate *date;
@property (retain) NSDate *primitiveDate;
@property NSInteger processID;

@end
```

Listing 8-3 Listing of the implementation of the Run class

```
#import "Run.h"

@implementation Run

@dynamic date, primitiveDate;

- (void) awakeFromInsert {
    // set date to now
    self.primitiveDate = [NSDate date];
}


- (int)processID {
    [self willAccessValueForKey:@"processID"];
    NSInteger pid = processID;
    [self didAccessValueForKey:@"processID"];
    return pid;
}


- (void)setProcessID:(int)newProcessID {
    [self willChangeValueForKey:@"processID"];
    processID = newProcessID;
    [self didChangeValueForKey:@"processID"];
}


- (void)setNilValueForKey:(NSString *)key {
    if ([key isEqualToString:@"processID"]) {
        self.processID = 0;
```

```
    }
    else {
        [super setNilValueForKey:key];
    }
}

@end
```

Document Revision History

This table describes the changes to *Core Data Utility Tutorial*.

Date	Notes
2009-03-04	Updated title to better reflect purpose.
2008-11-19	Updated to use garbage collection.
2008-10-15	Corrected implementation of applicationLogDirectory() in the complete source listings.
2008-02-08	Updated for Mac OS X v10.5.
2006-11-07	Corrected function name in final code listing.
2006-10-03	Corrected typographical errors.
2006-05-23	Added a warning about prerequisite requirements.
2006-04-04	Fixed minor typographical errors.
2005-12-06	Corrected minor errors in Complete Code Listing in "Creating the Managed Object Model."
2005-11-09	Corrected leak in sample code complete listing.
2005-10-04	Corrected use of NSCalendarDate as instance variable.
2005-08-11	Corrected memory leak in the managedObjectContext function code example.
2005-07-07	Corrected various minor typographical errors. Set application store location to ~/Library/Logs.
2005-04-29	New document that describes the creation of a low-level Core Data command-line utility.

Document Revision History