Core Data: The Basics

TN Valley Apple Developers
April 7, 2011 / May 5, 2011

What is Core Data?

- Core Data is a very important subframework of the Cocoa framework that provides an ORM for Objective-C and SQLite
  - ORM stands for Object-Relational Mapping, which is an object-oriented technique of handling database interactions

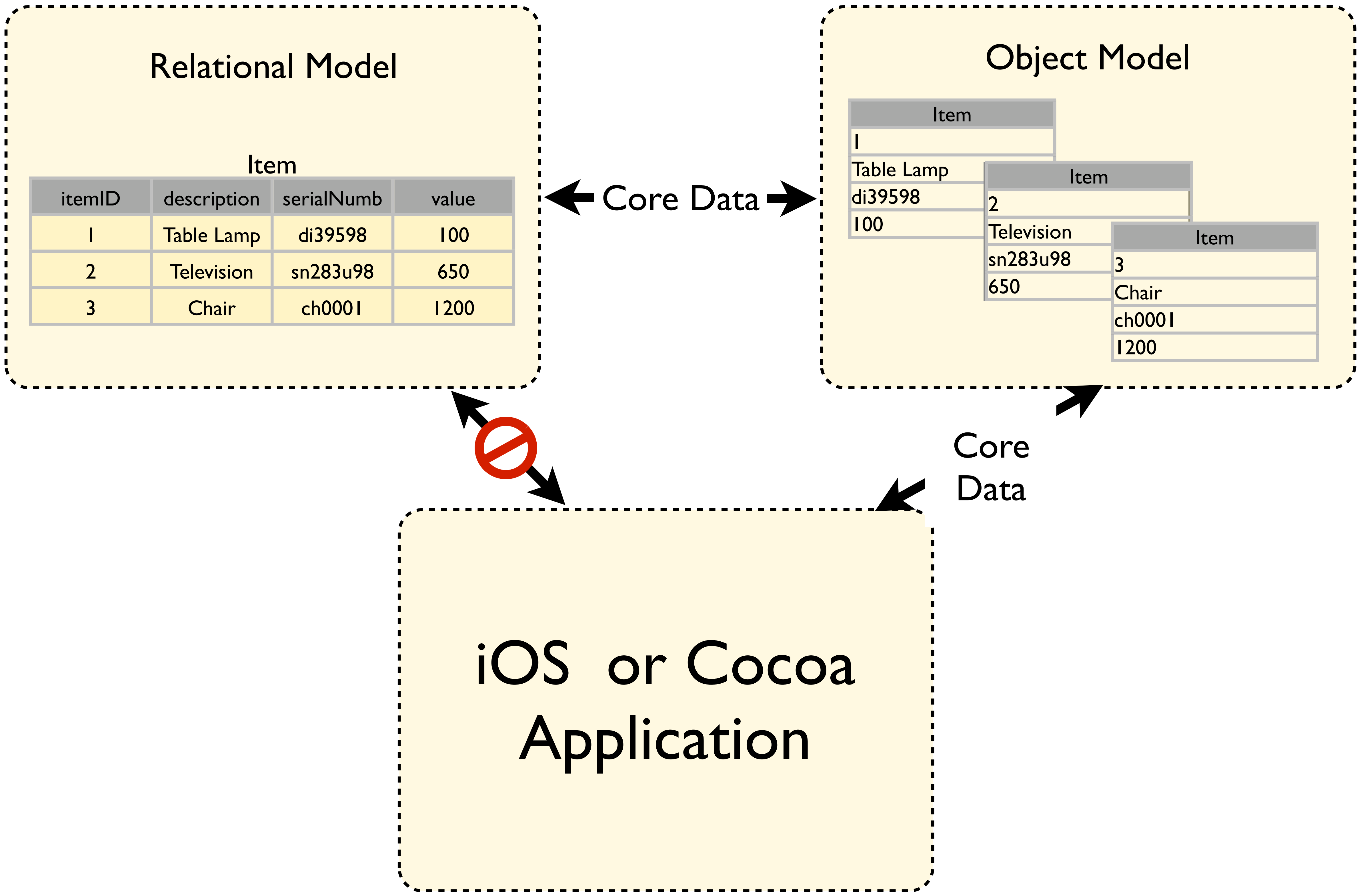
Object-Oriented Database?

- In simple terms, a class would (more or less) represent a table in a database
- A data object would represent a record in a database (row in a database table)
- An instance variable in that data object would represent a field in that record (or a column in the database table)

Core Data’s Role

- Core Data presents the records from the database to your application as an “object”
  - When these objects are edited, Core Data is responsible for updating the appropriate records in the database tables
  - Core Data is also responsible for creating new records and deleting old ones as objects are created and destroyed

Core Data’s Role



Relational Model

Item

itemID	description	serialNumb	value
1	Table Lamp	di39598	100
2	Television	sn283u98	650
3	Chair	ch0001	1200

Object Model

Item
1
Table Lamp
di39598
100

Item
2
Television
sn283u98
650

Item
3
Chair
ch0001
1200

Components

- The Class (or table) is called an **entity**
- A column (or instance variable) is called an **attribute**
- A key join between tables is called a **relationship**
- Both attributes and relationships are represented by “properties” in the data model class

More Components

- A **fetched property** is a query on a single field within a single managed object
  - Example: sameDepartment – returns all employees within a single department
- A **fetch request** is a class method that implements a canned query
  - Example: iPhoneProgrammer – returns all programmers that can do iOS development

NSManagedObjectModel

- The Objective-C class that represents the data object model in memory
- Manages access to all data stores for your application

NSPersistentStore-Coordinator

- Handles communications calls from different classes that trigger data access (read/writes)
- Manages data access to prevent data access collisions and minimize data corruption opportunities

Data Stores

- Default data store is SQLite (NSSQLiteStoreType)
- Also available is a binary flat file (NSBinaryStoreType)
- And a memory store (NSInMemoryStoreType)

Entities and Managed Objects

- Entities define the structure of your data
  - Every entity is an instance of the NSManagedObject class
- Managed objects are instances of data that conform to a defined Entity

Key-Value Coding

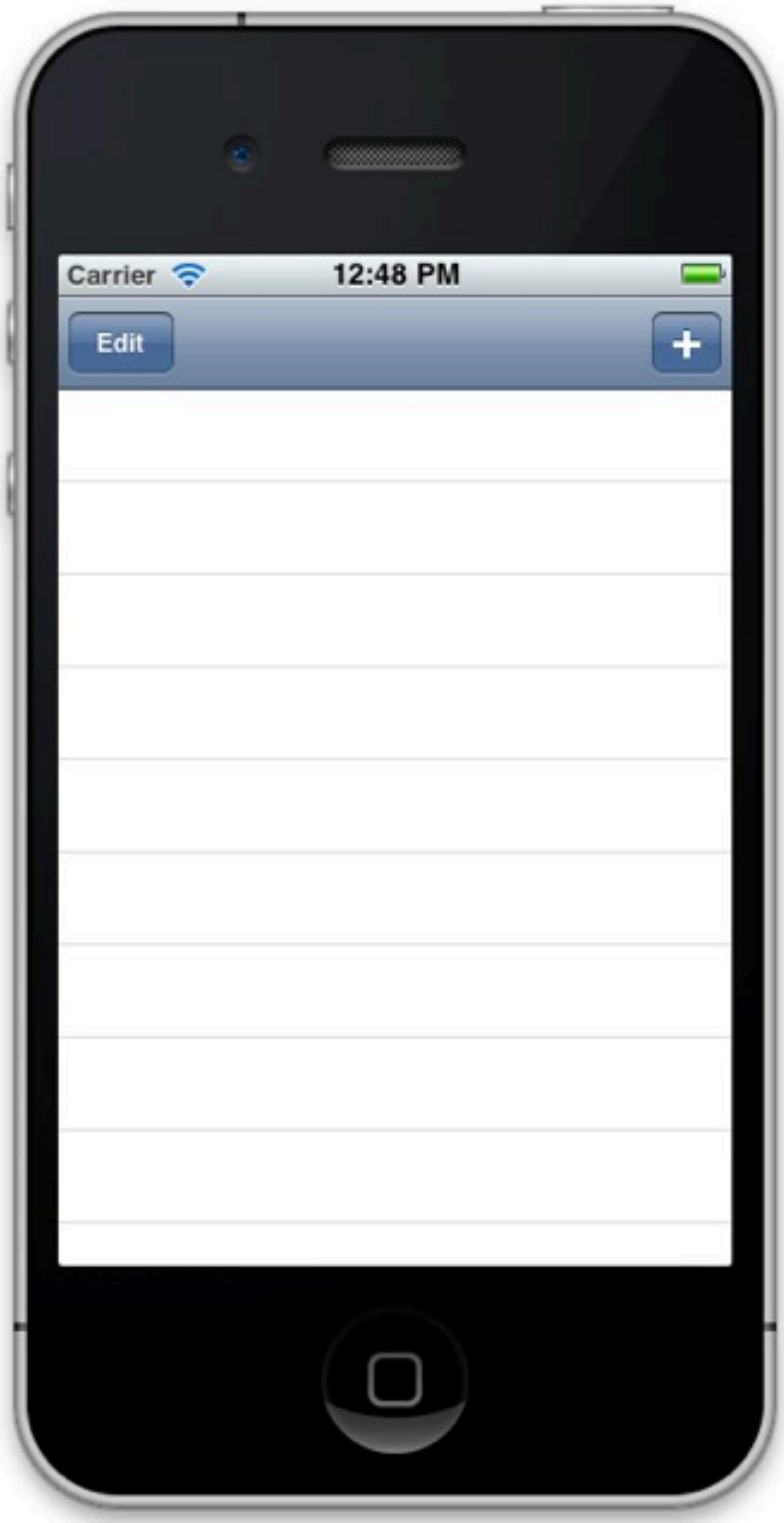
- NSManagedObject instances (entities) support key/value methods for accessing unique instances of data
- Very similar to the NSDictionary class

Managed Object Context

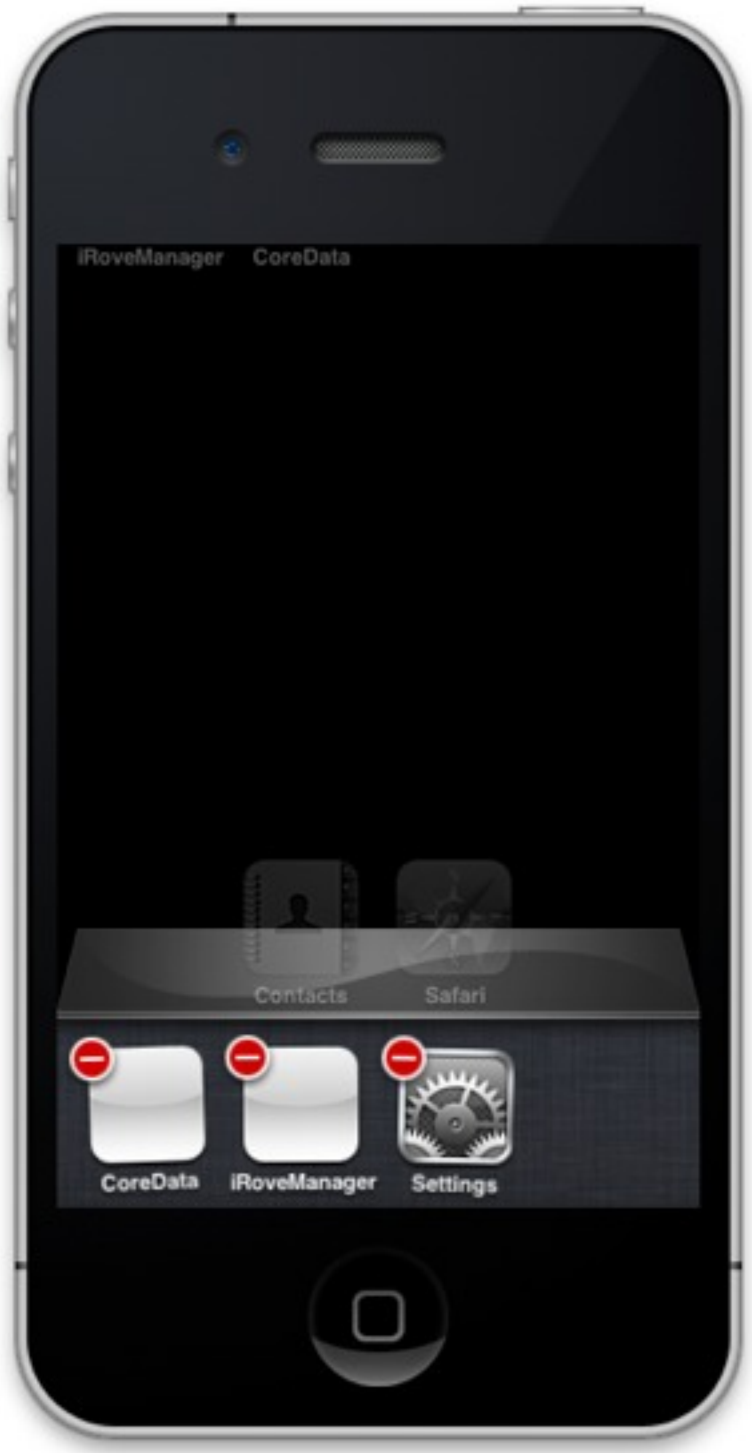
- Serves as the “gateway” between your entities and the rest of Core Data
  - Maintains state of all managed objects
  - Tracks changes to managed objects
  - Coordinates data changes with the persistent store coordinator

New Project

- In Xcode, create a new Navigation-based Application, making sure you check “Use Core Data” on the second screen (where you name the project). Call the project “CoreData”.
- Build and run your application



- If you press the "+" button, it will insert a new row showing the date / time the button was pressed.
- The "Edit" button can be used to delete any rows.



- Hit the Stop button in Xcode, but keep the simulator on the screen.
- Add a couple of rows to the table
- Hit the home button to send the program into the background
- Double click on the home button to see all programs running in the background
- Click and hold the "CoreData" icon until the icons start to jiggle on the screen, then click the red "-" button in the left-hand corner of "CoreData"– this quits the program completely. Click the "home" button again.
- Click the "CoreData" icon to fire it up again

Note: Your previously created data is still there. CoreData has saved the data to a persistent data store, where it can be recalled and used again.

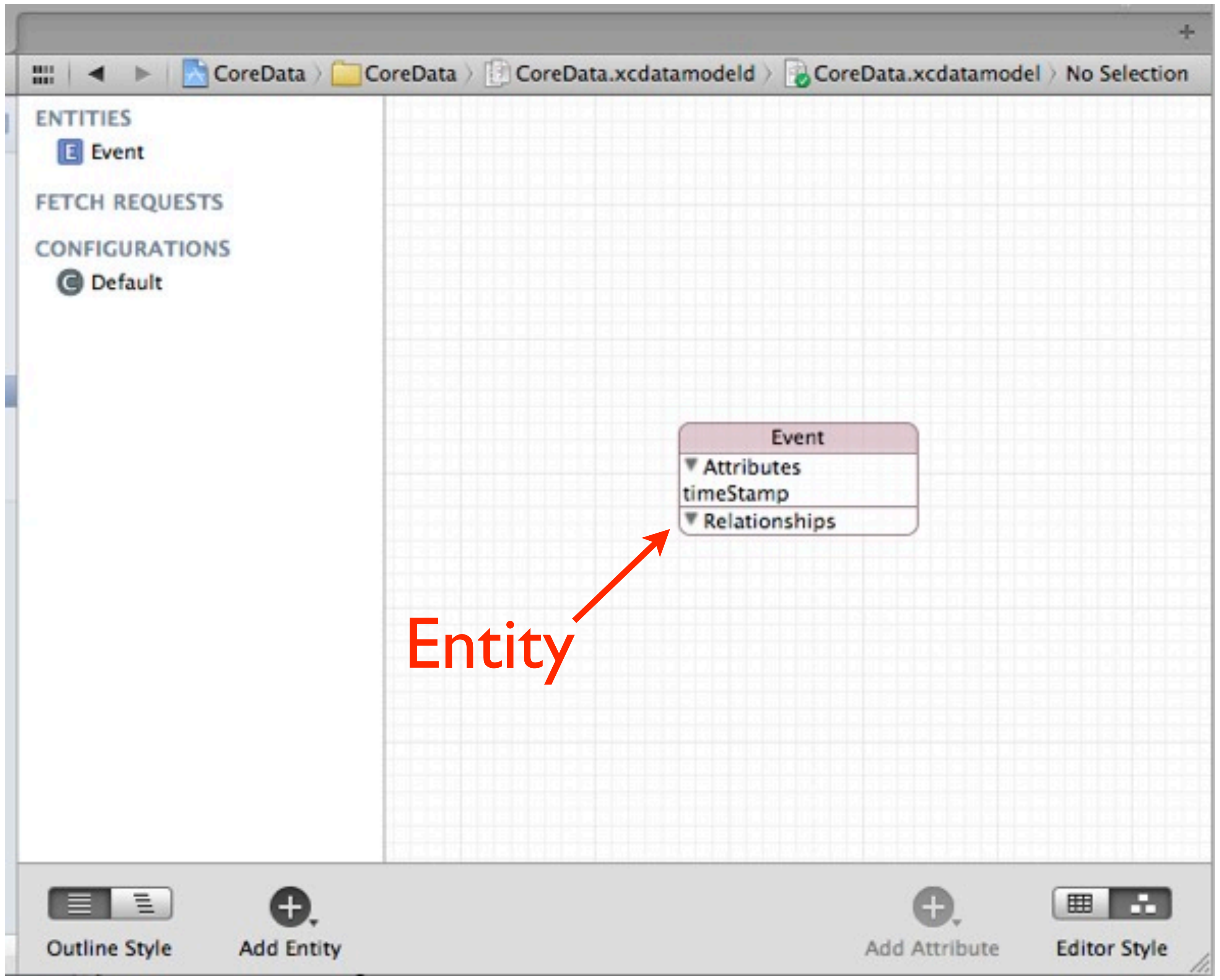
Core Data Concepts

- Persistent Store
- Data Model
- Persistent Store Coordinator
- Managed Object Context
  - Entities
  - Managed Objects
- Fetch request
  - Predicates

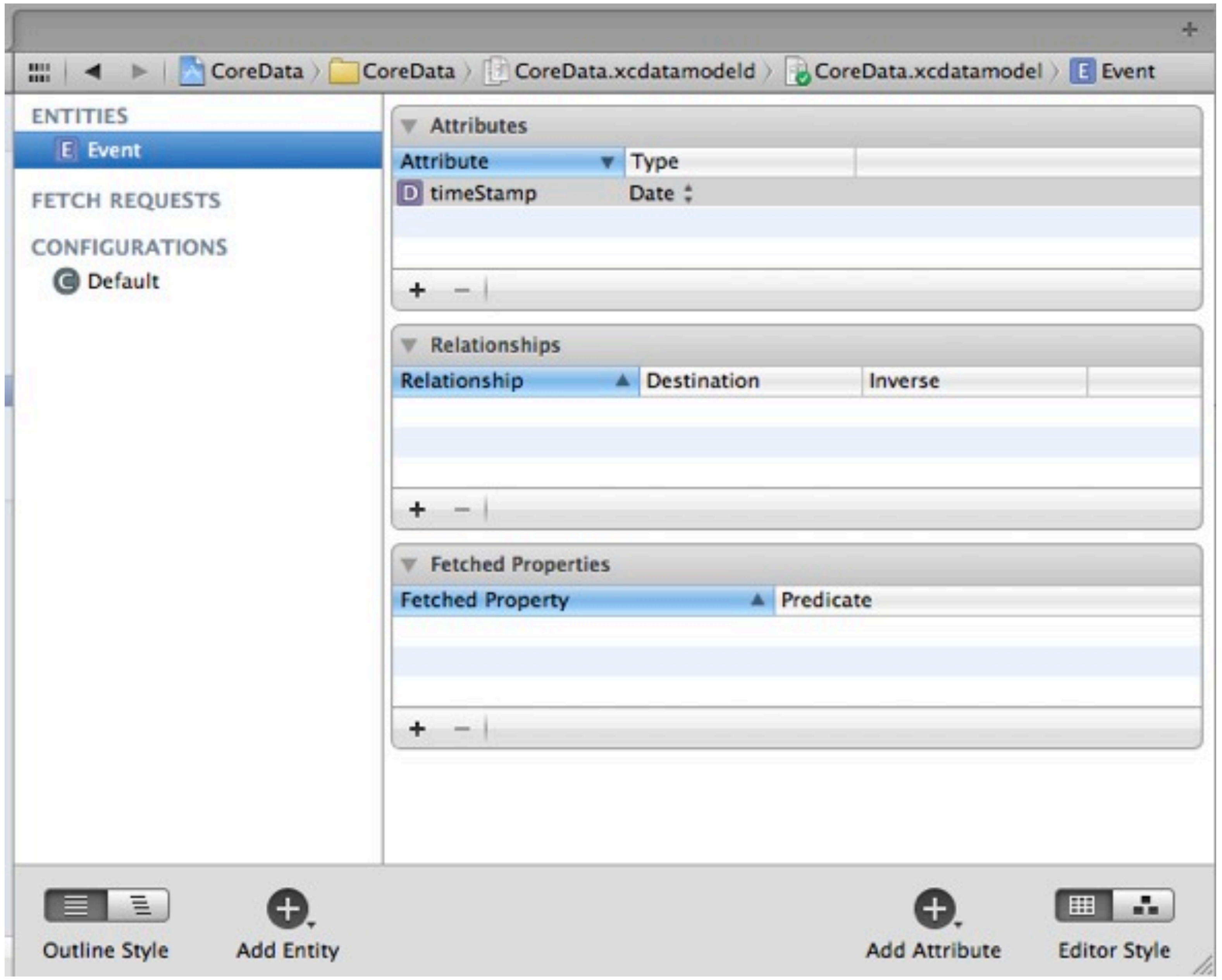
Data Model / Persistent Store

- The Persistent Store is where Core Data stores its data
- Every persistent store is associated with a single Data Model
  - The data model defines the types of data that can be stored in its associated persistent store

- In Xcode, open the Project Navigator
- Open the file called "CoreData. xcdatamodel
- Select the "Graph" Editor style button (lower right corner)



- Select the "Table" Editor button
- We have a single attribute, "timeStamp" of type "Date"
- This means we have a table (entity) with a single column (attribute) that stores a Date (data type)
- We currently have no defined relationships (table joins) or fetched properties (stored queries)



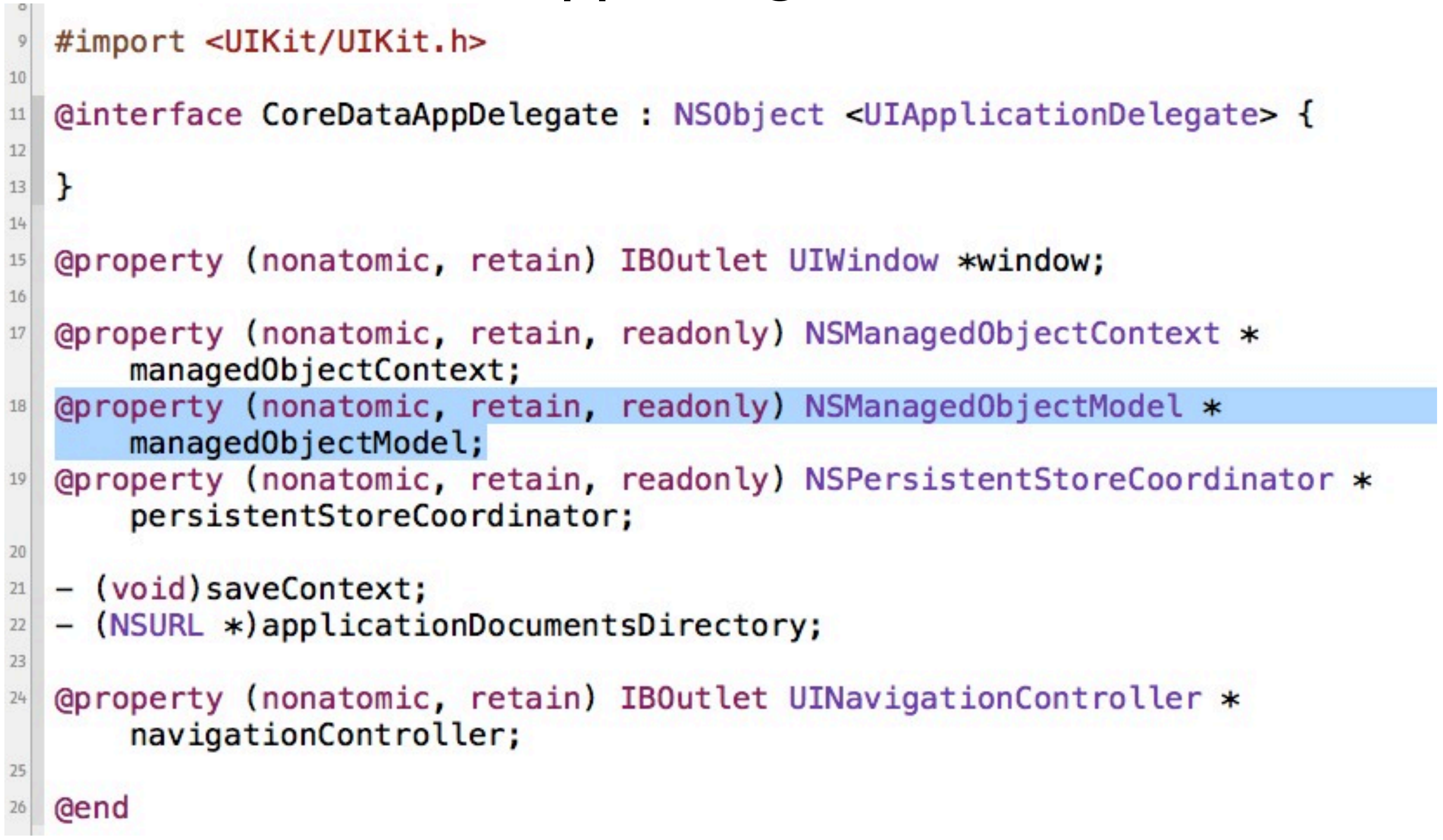
Attribute	Type
timeStamp	Date

Relationship	Destination	Inverse

Fetched Property	Predicate

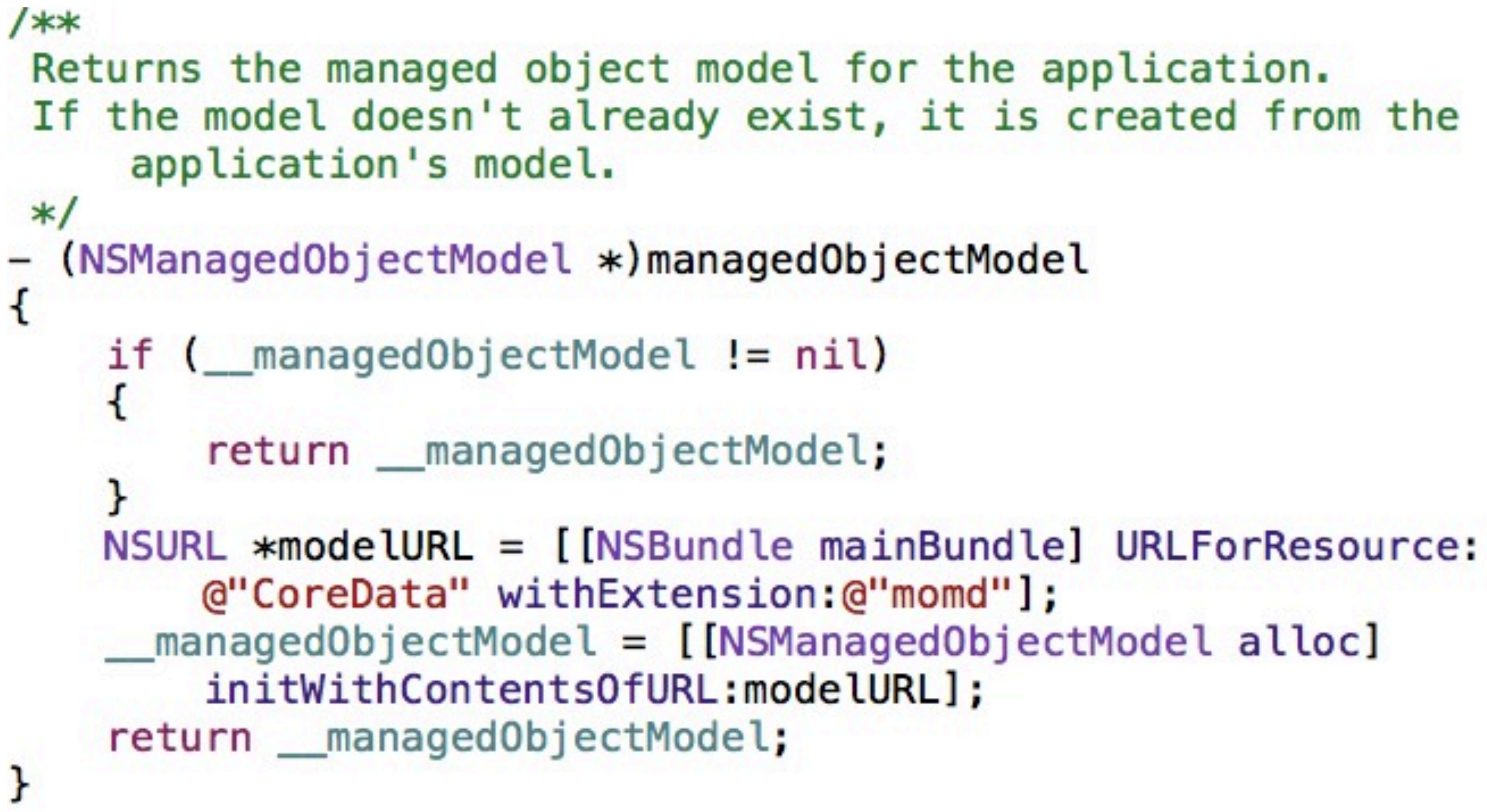
NSManagedObjectModel

- In the Project Navigator, select the “CoreDataAppDelegate” header file



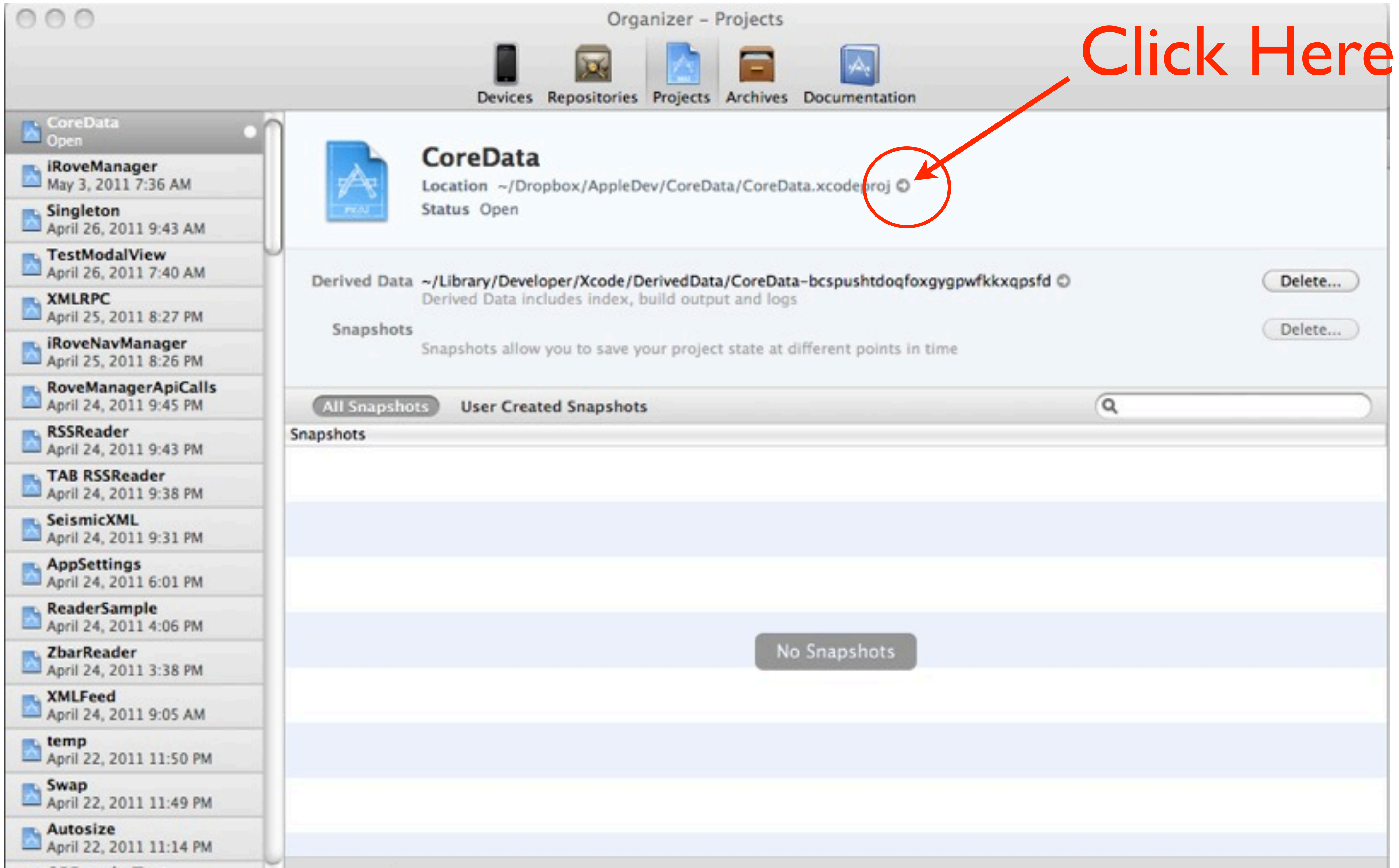
```
#import <UIKit/UIKit.h>

@interface CoreDataAppDelegate : NSObject <UIApplicationDelegate> {

}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@property (nonatomic, retain, readonly) NSManagedObjectContext *
    managedObjectContext;
@property (nonatomic, retain, readonly) NSManagedObjectModel *
    managedObjectModel;
@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator *
    persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)applicationDocumentsDirectory;

@property (nonatomic, retain) IBOutlet UINavigationController *
    navigationController;

@end
```

CoreDataAppDelegate.m



```
/**
 Returns the managed object model for the application.
 If the model doesn't already exist, it is created from the
     application's model.
 */
- (NSManagedObjectModel *)managedObjectModel
{
    if (__managedObjectModel != nil)
    {
        return __managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:
        @"CoreData" withExtension:@"momd"];
    __managedObjectModel = [[NSManagedObjectModel alloc]
        initWithContentsOfURL:modelURL];
    return __managedObjectModel;
}
```

- In Xcode, open the "Organizer" window and select "Projects"
- Pick the CoreData project



CoreData
Location ~/Dropbox/AppleDev/CoreData/CoreData.xcodeproj
Status Open

Derived Data ~/Library/Developer/Xcode/DerivedData/CoreData-bcspushtdoqfoxgygpwfkkxqpsfd
Derived Data includes index, build output and logs

Snapshots
Snapshots allow you to save your project state at different points in time

Snapshots

CoreData	Open
iRoveManager	May 3, 2011 7:36 AM
Singleton	April 26, 2011 9:43 AM
TestModalView	April 26, 2011 7:40 AM
XMLRPC	April 25, 2011 8:27 PM
iRoveNavManager	April 25, 2011 8:26 PM
RoveManagerApiCalls	April 24, 2011 9:45 PM
RSSReader	April 24, 2011 9:43 PM
TAB RSSReader	April 24, 2011 9:38 PM
SeismicXML	April 24, 2011 9:31 PM
AppSettings	April 24, 2011 6:01 PM
ReaderSample	April 24, 2011 4:06 PM
ZbarReader	April 24, 2011 3:38 PM
XMLFeed	April 24, 2011 9:05 AM
temp	April 22, 2011 11:50 PM
Swap	April 22, 2011 11:49 PM
Autosize	April 22, 2011 11:14 PM

CoreData Managed Object Model

- In the Finder window, click "Build > Products > Debug-iphonesimulator"
- Right-click on "CoreData.app" and select "Show Package Contents"
- The contents of the folder "CoreData.momd" (called a "resource bundle") is what gets used by "CoreDataAppDelegate::managedObjectModel" to create the CoreData managed object instance

managedObjectModel method

- This is the "accessor" method that is created for you that you should use to access the managedObjectModel instance.
- Never access the managedObjectModel directly – iOS uses a method called "lazy loading" to wait until you actually need to access the managedObjectModel instance before it creates it. Trying to access it directly can cause segmentation faults at runtime because it may not exist in memory when you try to access it.
- Always use the "managedObjectModel" method to return an instance of the managedObjectModel to to work with.

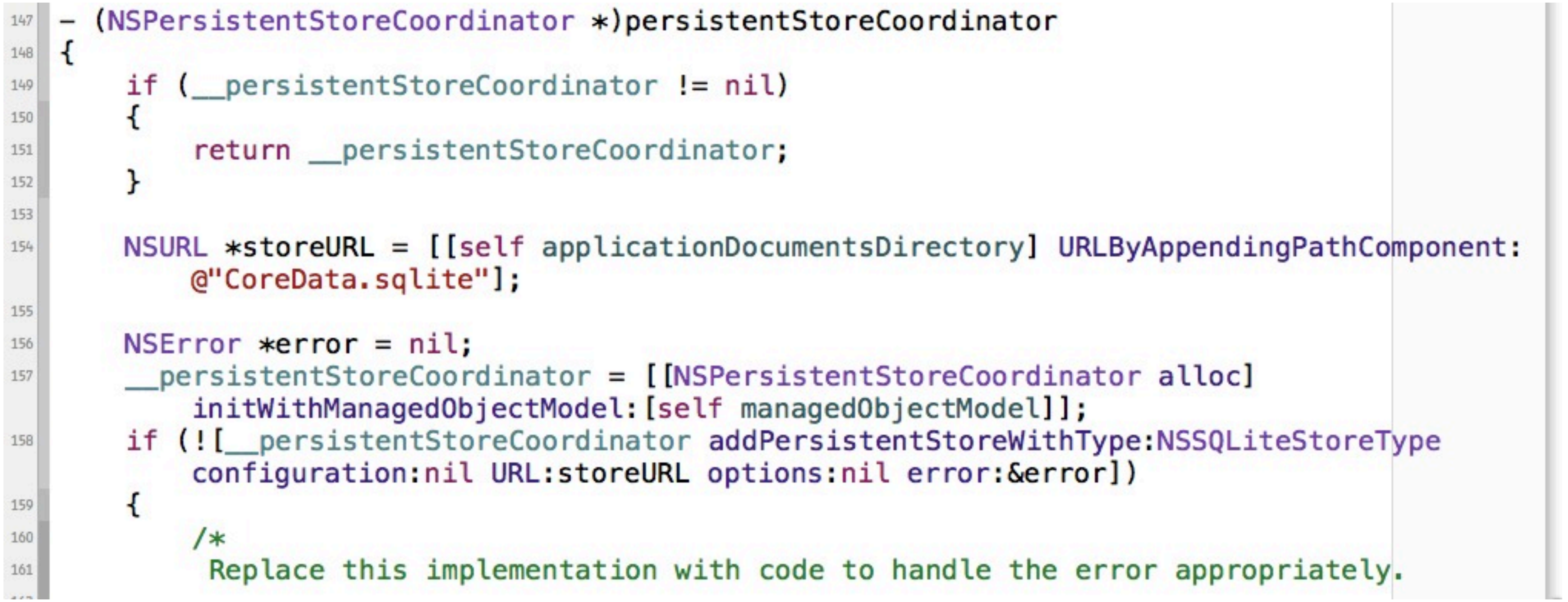
Data Entities

- XCode does support multiple data entities (tables)
- Most iOS projects will make use of a single table
- Multiple entities get combined into a single managedObjectModel

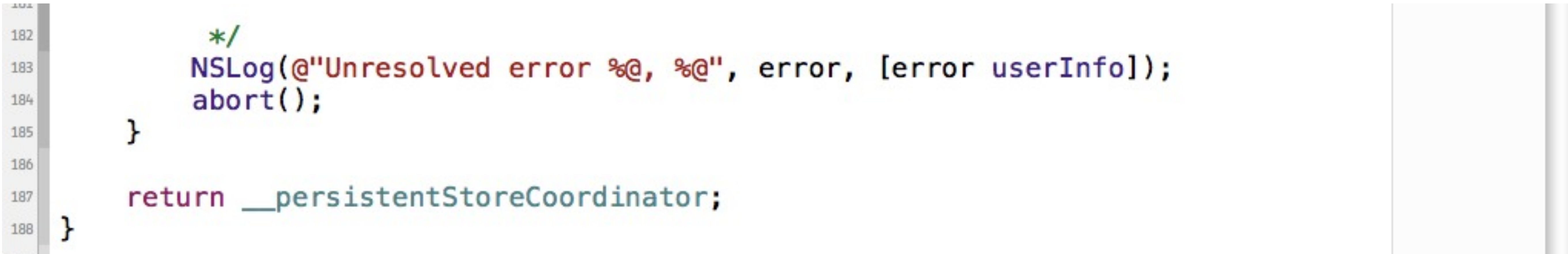
Persistent Store Coordinator

- The Persistent Store isn’t actually represented by an Objective-C class
- The Persistent Store Coordinator, and instance of NSPersistentStoreCoordinator, controls access to the persistent store
- The Persistent Store Coordinator manages all requests to access data to prevent conflicts and database locking, which can happen if you have multiple classes trying to access the data store at the same time

AppDelegate::persistent StoreCoordinator



```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
    if (__persistentStoreCoordinator != nil)
    {
        return __persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:
        @"CoreData.sqlite"];

    NSError *error = nil;
    __persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
        initWithManagedObjectModel:[self managedObjectModel]];
    if (![__persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
        configuration:nil URL:storeURL options:nil error:&error])
    {
        /*
         Replace this implementation with code to handle the error appropriately.
```



```
         */
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return __persistentStoreCoordinator;
}
```

AppDelegate::persistentStoreCoordinator

- As with the managedObjectModel method, the persistentStoreCoordinator method uses lazy-loading to create the persistent store object

CoreData.sqlite

- This is the database that is created to store your data
- The database is in the application's sandbox, in the "Documents" folder
- You can actually see the iPhone simulator's sandbox by navigating to:

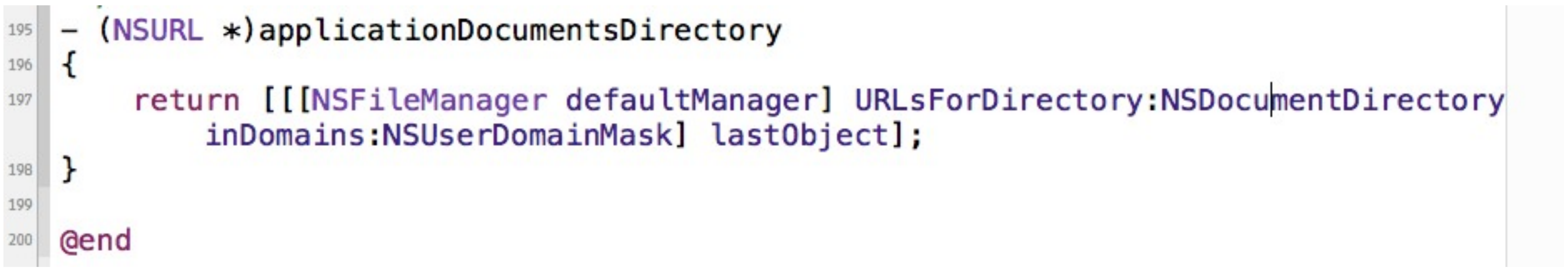
  "Library > Application Support > iPhone Simulator > 4.3.2 > Applications" and finding the GUID (**G**lobally **U**nique **ID**entifier) folder associated with your project

Firefox Plug-in

- Firefox has a plugin called SQLite Manager that lets you view and manipulate the contents of SQLite databases.
- USE WITH CAUTION: you can screw up your database if you inadvertently change something without realizing it.

AppDelegate::application DocumentsDirectory

- The applicationDocumentsDirectory method returns the location of the applications Documents directory for the current application.



```
- (NSURL *)applicationDocumentsDirectory
{
    return [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
        inDomains:NSUserDomainMask] lastObject];
}

@end
```