IOS

Introduction to Core Data: Your First Step to Persistent Data

ZIAD TAMIM 15TH DEC '12 84

Editor's note: *After we published the **tutorial about saving data in plist file**, some readers asked about Core Data and how we can use it to save persistent information. This week, we work with **Ziad Tamim**, an independent iOS developer, to give you an introduction of Core Data and work with you to build a sample app using Core Data.*

This tutorial talks about persistence on iPhone (or other iOS devices). What I me
that's in your apps stay around between application launches. Persistence lets u
retrieve it, so that users don't have to reenter all their data each time they use th
ways to store data in iOS devices but most of them aren't good enough to store a
used to save settings or to preload some data such as "Property List" and "Archi
through Core Data to see how you can utilize it to manage data in database.

The focus of the tutorial is to provide a practical introduction of Core Data frame
through our tutorials about **Storyboard** and **UITableView**. I will not give in-dept
controller in Storyboard but you can always refer to the **earlier tutorials** to gain

Core Data is not a Database

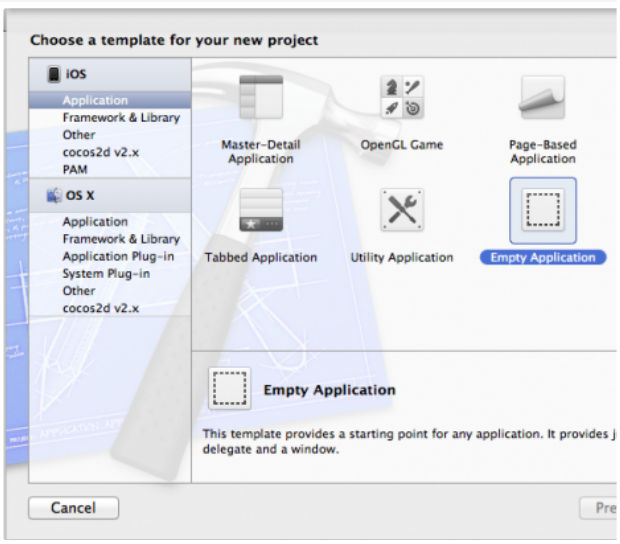
When we talk about persistent data, people probably think of database. If you are familiar with Oracle or MySQL, you know that relational database stores data in the form of table, row and column, and it usually facilitates access through what-so-called SQL query. However, don't mix up Core Data with database. Though **SQLite** database is the default persistent store for Core Data on iPhone, Core Data is not a relational database. It is actually a framework that lets developers store (or retrieve) data in database in an object-oriented way. With Core Data, you can easily map the objects in your apps to the table records in the database without even knowing any SQL.

To illustrate the concept, let's begin and create your first app using Core Data. This app is called **My Store**. It is a very simple app that stores all devices you have by collecting the **name**, **version**, **company**.

MyStore App using Core Data

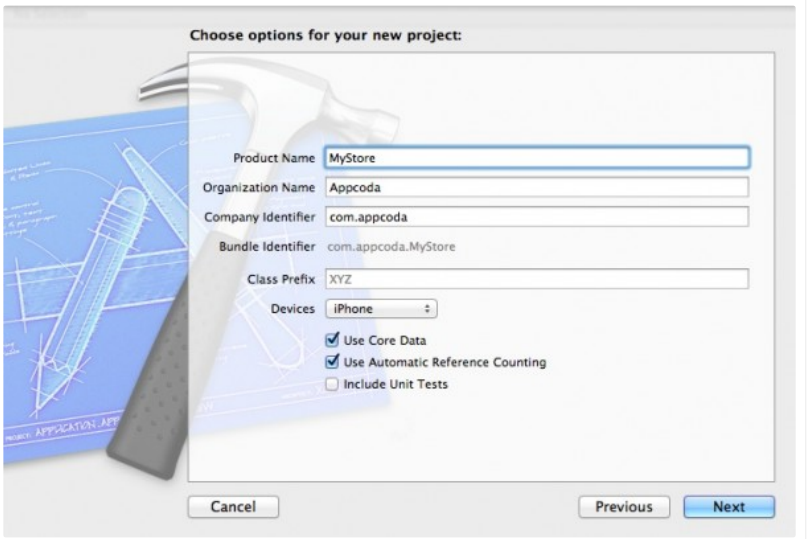
Creating a Sample App with Core Data

First let's create a project with Core Data. Open Xcode and create a new Project, as shown below.



Create a New Project with Empty Application Template

At the next screen, enter **MyStore** as a name of the project, select **iPhone** in Devices family and don't forget to select the options **Use Storyboards, Use Core Data, Use Automatic Reference Counting**. Press next and create.



Set up Xcode Project Options – Remember to select Use Core Data

Core Data Stack

Before we start working on the project, you first have to understand the Core Data Stack:

Managed Object Model – It describes the schema that you use in the app. If you have a database background, think of this as the database schema. However, the schema is represented by a collection of objects (also known as entities). In Xcode, the Managed Object Model is defined in a file with the extension ***.xcdatamodeld***. You can use the visual editor to define the entities and their attributes, as well as, relationships.

Persistent Store Coordinator – SQLite is the default persistent store in iOS. However, Core Data allows developers to setup multiple stores containing different entities. The Persistent Store Coordina different persistent object stores and save the objects to the stores. Forget about You'll not interact with Persistent Store Coordinator directly when using Core Da

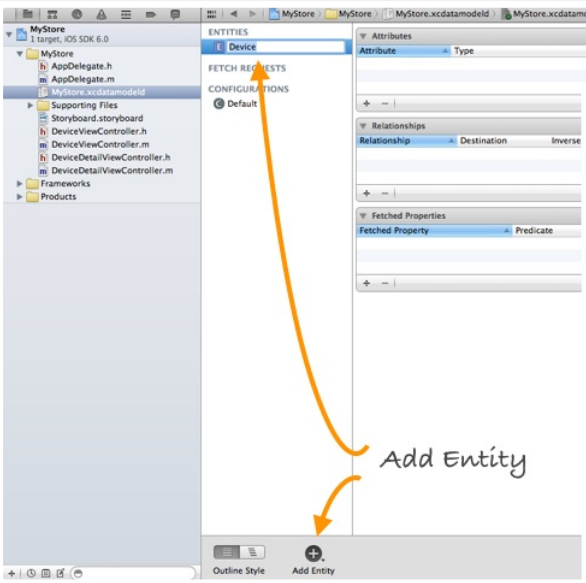
Managed Object Context – Think of it as a "scratch pad" containing objects that Its job to manage objects created and returned using Core Data. Among the co Managed Object Context is the one you'll work with for most of the time. In gen save objects in persistent store, the context is the first component you'll talk to.

The below illustration can probably give you a better idea about the Core Data St

Core Data Stack

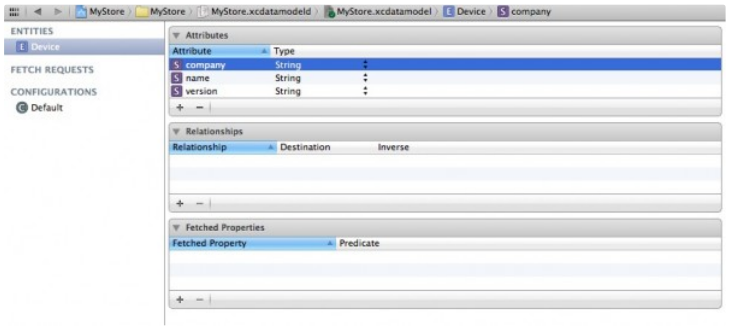
Defining Managed Object Model

Let's move on to build the app. The first step is to open the Data Model named ***MyStore.xcdatamodeld*** and define the object model. Here we'll define a Device entity that will be used to store the device information to database. To create an entity, click the + button in the bottom-left of the editor view and name the entity as **Device**.



Add Device entity in the model

Once you create a new entity, you need to add attributes to it. Click on the + butt
Add three attributes including name, version and company. Set the type as Strin

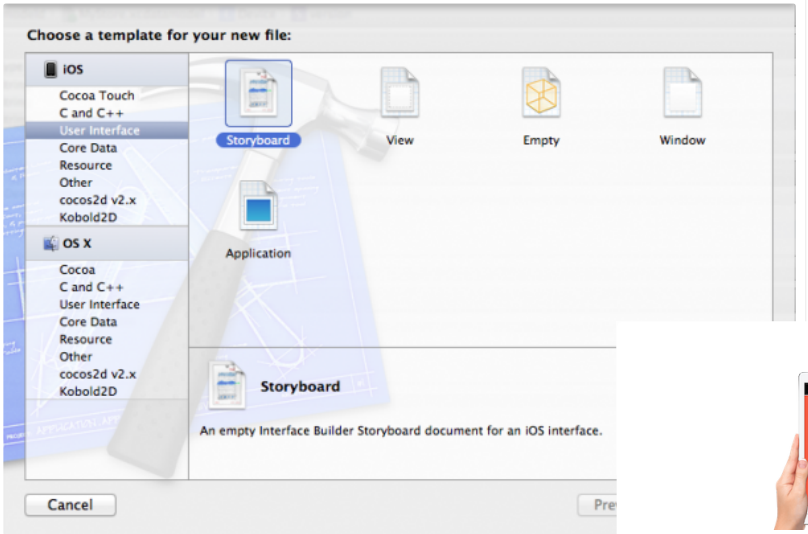


Add 3 Attributes (company, name and version) to the Device entity

Designing the User Interface

Note: While we encourage you to build the user interface, you can also skip the procedures and **download the project template from here**. The template already comes with the Storyboard and set up all the view controller classes for you. This gives you a good starting point to work on Core Data. If you use the template, you can skip this section and go directly to the "Diving Core Data" section.

The next thing we need to do is to create the Storyboard that defines the views of our app. Navigate to File > New > New File and choose Storyboard in the **User Interface** template. Click next and select the iPhone device family, click create.



Creating the Storyboard

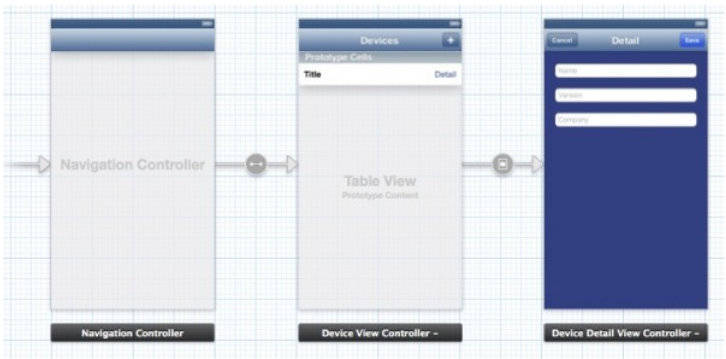
Once created, make sure to set the "Storyboard" you've just created as the main

Set the Storyboard you just created as the Main Storyboard

Also don't forget to delete all the generated code in the method *-(BOOL)application:application didFinishLaunchingWithOptions:launchOptions* inside the **AppDelegate** file. The method should be as simple as this:

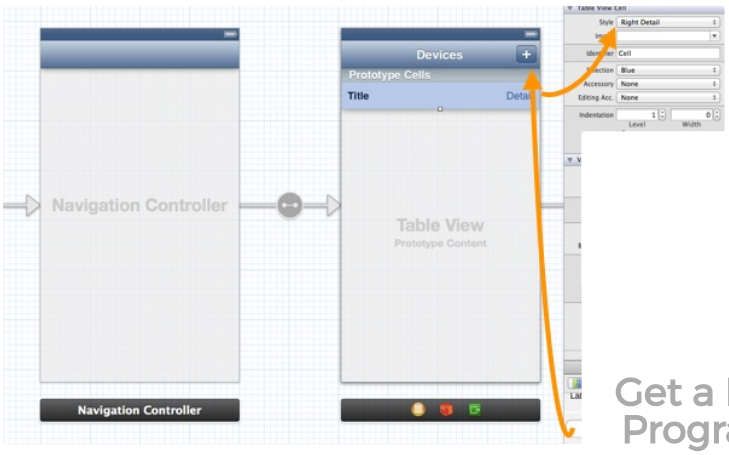
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launch
{
    return YES;
}
```


Go to Storyboard and create the user interface like below:



MyStore App – Storyboard

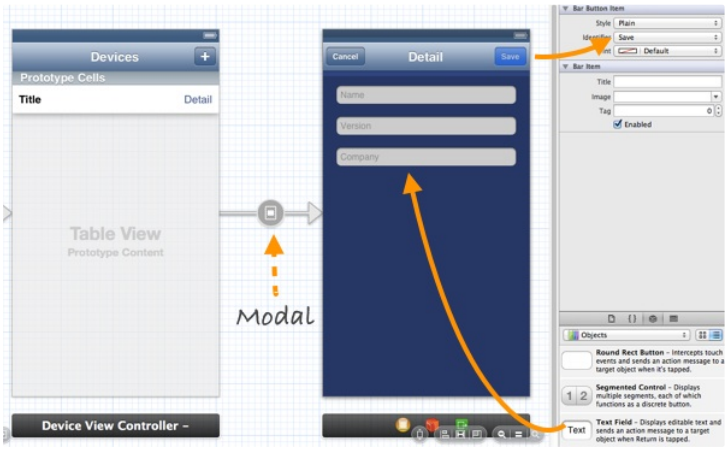
First, drag a Table View Controller and embed it in a Navigation Controller. Drag a button to the top-right part of navigation bar and set the identifier as "Add". This will automatically change the button to a "+" button. Next, select the prototype cell and change its style to "Right Detail".



Creating the Table View Controller

Drag a View Controller to the Storyboard and add a Navigation Bar to the top of t
the navigation bar. Name one as "Cancel" and the other one as "Save". In the cc
name the placeholder attributes as "Name", "Version" and "Company".

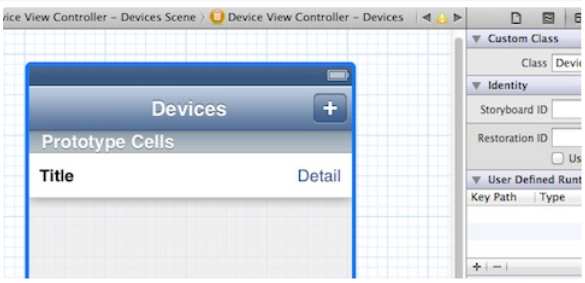
This detail view will be shown when user taps the "+" button in the table view c
Control key, click the "+" button and drag towards the detail view controller. Se
connect the table view controller and detail view controller.



Designing the Detail View Controller

Creating View Controller Classes

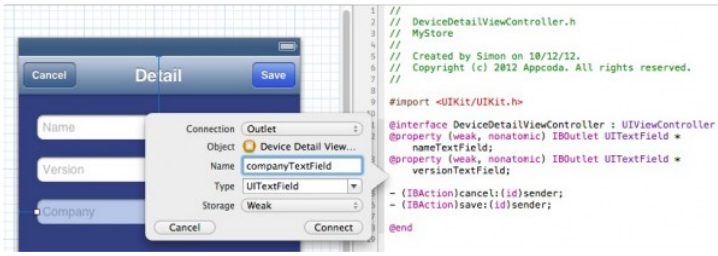
Create a new class by right-clicking on the **MyStore** folder > New File > Objective-C class, and name the class as **DeviceViewController**. Make it as a subclass of **UITableViewController**. Navigate to the Storyboard, select the Table View Controller and associate it with the DeviceViewController class.



Set the Custom Class as DeviceViewController

Once done, do the same steps to create a new class named **DeviceDetailViewCon** Storyboard and set the custom class of the detail view controller as the "DeviceD

Lastly, wire up the UITextFields to the **DeviceDetailViewController** header file a save and cancel buttons respectively.



Creating IBOutlet and Action Methods

Your code should like this:

```
@property (weak, nonatomic) IBOutlet UITextField *nameTextField;
@property (weak, nonatomic) IBOutlet UITextField *versionTextField;
@property (weak, nonatomic) IBOutlet UITextField *companyTextField;

- (IBAction)cancel:(id)sender;
- (IBAction)save:(id)sender;
```

Diving into Core Data

With the user interface, it's time to go into the details of Core Data. Apparently, there are a couple of areas we have to implement:

1. Save device information in the Detail View Controller
2. Fetch device information from persistent store (i.e. SQLite database) and populate the data into Table View Controller

We'll look into the implementation one by one.

Saving Device Information

First, we need to implement the **DeviceDetailViewController** to let user add the d

DeviceDetailViewController.m file and add the following code after @implemen

```
- (NSManagedObjectContext *)managedObjectContext {
    NSManagedObjectContext *context = nil;
    id delegate = [[UIApplication sharedApplication] delegate];
    if ([delegate performSelector:@selector(managedObjectContext)]) {
        context = [delegate managedObjectContext];
    }
    return context;
}
```

Recalled that we've selected the Core Data option when creating the project, Xco

object context in AppDelegate. This method allows us to retrieve the managed ob

Later we'll use the context to save the device data.

Next, we'll implement the "save" and "cancel", add the necessary code to look l

```
- (IBAction)cancel:(id)sender {
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (IBAction)save:(id)sender {
    NSManagedObjectContext *context = [self managedObjectContext];

    // Create a new managed object
    NSManagedObject *newDevice = [NSEntityDescription insertNewObjectForEntityForName:@"Device" inMana
    [newDevice setValue:self.nameTextField.text forKey:@"name"];
    [newDevice setValue:self.versionTextField.text forKey:@"version"];
    [newDevice setValue:self.companyTextField.text forKey:@"company"];

    NSError *error = nil;
    // Save the object to persistent store
    if (![context save:&error]) {
        NSLog(@"Can't Save! %@ %@", error, [error localizedDescription]);
    }

    [self dismissViewControllerAnimated:YES completion:nil];
}
```

When user taps the "Cancel" button, we expect the app to close the detail view controller. Line 2 of the above code invokes the dismissViewControllerAnimated method to dismiss the current view controller with animation.

For the "save" method, we first grab the managed object context. Every object that Core Data stores is inherited from NSManagedObject. So we first create a new instance of NSManagedObject for the "Device" entity that we've defined in the object model. NSEntityDescription class provides a method named "insertNewObjectForEntityForName" for developer to create a managed object. Once you created the managed object (i.e. newDevice), you can set the attributes (name, version, company) using the user input. Lastly, we call up the "save:" method of the context to save the object into database.

You can now hit the Run button to try out your app. Tap the "+" button to bring up the Detail View and save a new device. However, the new device is not yet displayed in the table. Let's move on to see how you can fetch the device information from database.

Fetching Device Information

Open **DeviceViewController.m**, add a "devices" property to it so we can save all

```
@interface DeviceViewController ()
@property (strong) NSMutableArray *devices;

@end
```

Again, add the following code after "@implementation DeviceViewController" fo
context:

```
- (NSManagedObjectContext *)managedObjectContext
{
    NSManagedObjectContext *context = nil;
    id delegate = [[UIApplication sharedApplication] delegate];
    if ([delegate performSelector:@selector(managedObjectContext)]) {
```

```
        context = [delegate managedObjectContext];
    }
    return context;
}
```


Next, add a viewDidAppear method:

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    // Fetch the devices from persistent data store
    NSManagedObjectContext *managedObjectContext = [self managedObjectContext];
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName:@"Device"];
    self.devices = [[managedObjectContext executeFetchRequest:fetchRequest error:nil] mutableCopy];

    [self.tableView reloadData];
}
```

Like what we've done in the Detail View Controller, we first grab the managed object context. To fetch device information from database, the code above creates a new instance of **NSFetchRequest** and set the entity **Device** and invokes "executeFetchRequest" method to retrieve all the devices from the database. If you are familiar with relational databases, this instance works like the SELECT clause.

> **Note:** If you're new to viewDidAppear method, it is a method that will be called automatically every time a view is displayed. It's unlike the viewDidLoad method that is invoked once when the controller is loaded.

Populating Device Information into Table View

As we would like to display these data into the table view we need to implement the data source of it, to do that add the below code:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection
{
    // Return the number of rows in the section.
    return self.devices.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtI
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentif

    // Configure the cell...
    NSManagedObject *device = [self.devices objectAtIndex:indexPath.r
    [cell.textLabel setText:[NSString stringWithFormat:@"%@ %@", [dev
    [cell.detailTextLabel setText:[device valueForKey:@"company"]];

    return cell;
}
```

If you have used UITableViewController before, the code above is the simple way to display data into the table view. If you check the code you will notice the **NSMangedObject** is pretty much like NSDictionary. It gathers all the attributes of the entity (i.e. Device) and you can simply use the "valueForKey" method to grab the attribute value.

Note: If you're new to UITableView, you can check out the earlier tutorials about UITableView.

That's it. Let's try to run the app and test it. If everything is okay, your app should like this. Try to add some devices and the device information should be populated automatically in the table view.

MyStore App using Core Data

What's Coming Next

It's a lengthy tutorial but we try to elaborate the implementation as detail as pos[illegible] can see, with Core Data, you don't have to care about SQL to save and retrieve da behind the scene. This tutorial kicks off the first part of Core Data series. Later v relationship.

Lastly, let me end the tutorial with an exercise. Try to complete the app by addir and delete an existing device by selecting a row in the table view.

Hope you enjoy the tutorial and feel to leave us comment.

Update: Check out **part 2 of the Core Data tutorial series**!

BEGINNER CORE DATA DATABASE PROGRAMMING IOS PROGRAMMING OBJECTIVE C

Ziad Tamim

Ziad is a senior iOS Developer, Mobile Strategy Advisor and Consultant for startups. He has been writing iOS apps and games since the infancy of the App Store and built many apps for clients. Right now, he runs a mobile development studio called IntensifyStudio. You can contact him via Twitter.

RELATED ARTICLES

IOS

How To Create UIPageViewController Using Storyboard

IOS

Swift Tutorial: Building an iOS Chat App Using Socket.IO

IOS

How To Migrate from Parse to Self-hosted MongoDB

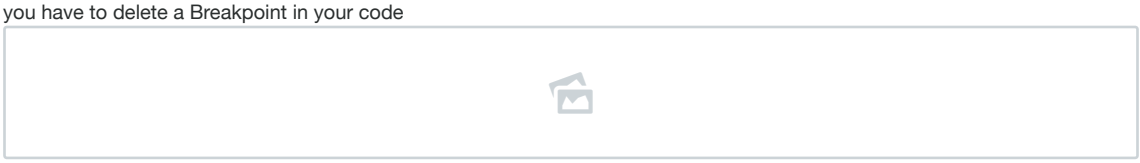
{@autoreleasepool {

return UIApplicationMain(argc, argv, nil, NSStringFromClass([myStoreAppDelegate class

]));

}
}
10 · Reply · Share ›

wahiba ➚ jadacoast • 3 years ago
you have to delete a Breakpoint in your code



· Reply · Share ›

ish3lan ➚ wahiba • 3 years ago
in DeviceDetailViewController:

add .text after calling uitextfields :

[newDevice setValue:self.nameTextField.text forKey:@"name"];

[newDevice setValue:self.versionTextField.text forKey:@"version"];

[newDevice setValue:self.companyTextField.text forKey:@"company"];

as we should give a NSString to be saved not a UITextField

Hope that will help

:)
· Reply · Share ›

Nathaniel Lee ➚ jadacoast • 4 years ago
You still there?
· Reply · Share ›

Visitor • 4 years ago
In, DeviceViewController.m, method:

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPathNo visible @interface for 'UITableView' declares the selector

'dequeueReusableCellWithIdentifier:forIndexPath:'

Why it is giving me an ARC issue?
8 · Reply · Share ›

Chavira Romero ➚ Visitor • 4 years ago
The problem that you're having is that you didn't set the identifier of the Table Vi
the screenshot:

Greetings!
5 · Reply · Share ›

Guest ➚ Chavira Romero • 4 years ago
Aha! Thats the issue that Ihad too. Thanks for this :-)
1 · Reply · Share ›

Marco Rosas ➚ Chavira Romero • 4 years ago
Yeah, the same problem I faced... thanks for this!
· Reply · Share ›

Tien Dung. ➚ Chavira Romero • 4 years ago
That's right, it's maybe a missing.
P/S: Add this into tutorial bro !!!
· Reply · Share ›

Alexandru ➚ Visitor • 4 months ago
maybe you try to call the function on the wrong object?
Reply • Share ›

Robert • 4 years ago
mistake?

DeviceDetailViewControllerUIViewController. Again, go to Storyboard and set the custom class of the detail view controller as the "DeviceDetailViewController".

should read?

DeviceDetailViewController. Make it as a subclass of UITableViewController. Again, go to Storyboard and set the custom class of the detail view controller as the "DeviceDetailViewController".
6 Reply • Share ›

Cristiano72 • 4 years ago
i have this problem
4 Reply • Share ›

aradddd ➚ Cristiano72 • 3 years ago
I have the same problem, please help us
Reply • Share ›

scud • 4 years ago
I'm getting a SIGABRT error when I try to build at:
UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

in the DeviceViewController.m file. Can anyone help?
4 Reply • Share ›

Juan Nozaleda ➚ scud • 3 years ago
Change the cell style into left detail
Reply • Share ›

Priyal Jain ➚ Juan Nozaleda • 2 years ago
Whats the difference between left/right Detail ?
Reply • Share ›

B • 4 years ago
#import <uikit uikit.h="">
#import "AppDelegate.h"
int main(int argc, char *argv[])
{
@autoreleasepool {
return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
}
} --Got a Thread 1:signal SIGABRT - program crashed after i tried to hit save. I have an
to programming....please any advice??
4 Reply • Share ›

Jason ➚ B • 4 years ago
Getting the same error as well. Felt like I learned quite a bit but stuck with this ru
1 Reply • Share ›

Richard ➚ B • 4 years ago
I'm getting the same error too. Can't find what I seem to be doing wrong.
1 Reply • Share ›

Juan Nozaleda ➚ Richard • 3 years ago
Change the Cell Style into left detail
Reply • Share ›

Priyal Jain ➚ Juan Nozaleda • 2 years ago
It isn't working
Reply • Share ›

Nathaniel Lee ➚ B • 4 years ago

Can you please post what it says in the console?

Reply • Share ›

This comment was deleted.

Priyal Jain ➚ Guest • 2 years ago

I had the Same issue :/ but i resolved it. Its been 2 years since your comment, you would have resolved it surely but I am posting my edits for others reference.

Edit:
1. re build the connection between the add button and DeviceDetailsViewController and ensure that it is modally connected.
2. DeviceDetailsViewController should be a subclass of ViewController. This is not clear from the tutorial.
3. I added a method without any body to DeviceViewController.m:
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

}

It worked fine. Hope it helps.

Reply • Share ›

Curlypaws • 4 years ago

This is really good - but in trying to add update and delete capabilities I'm struggling to find out how I'd identify the correct record and then update/delete it. I'm guessing that I need to pass the context. Should I be looking at NSFetchedResultsController?

2 Reply • Share ›

Simon Ng Mod ➚ Curlypaws • 4 years ago

You can check out the new tutorial:

http://www.appcoda.com/core...

Reply • Share ›

sankar ram • 3 years ago

Please help me in this problem!! I am getting error in this line in DeviceViewController.m file

UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:indexPath];

1 Reply • Share ›

Juan Nozaleda ➚ sankar ram • 3 years ago

Try to change the cell style. put left detail

Reply • Share ›

jacob banks • 4 years ago

When i run it it is saying Application windows are expected to have a root view controlle my tableview just a white screen

1 Reply • Share ›

RaviKumar Yaganti ➚ jacob banks • 2 years ago

you have to remove the default code in the didfinishlaunchingwithoption in appc

Reply • Share ›

jacob banks ➚ jacob banks • 4 years ago

can anyone help with that?

Reply • Share ›

Ashish Porecha • 4 years ago

Under Persistent Store Coordinator is a line:

Forget about it you don't understand what it is.

WTF?!?! HAHAHAHAHAHA!!!

1 · Reply · Share ›

Pakap84 • 4 years ago
Great tutorial but, when I use the Iphone simulator, i have a dark screen (nothing displayed). For xcode, there is no errors...
1 · Reply · Share ›

iamxyt ➚ Pakap84 • 4 years ago
Setting "Main storyboard file base name" information in the MyStore-info.plist file, which tells the Xcode which storyboard you are using,
5 · Reply · Share ›

Pakap84 • 4 years ago
Great tutorial but I've a big problem. At the end, when I want to run my application (i've a dark screen...). Nothing is display and there are no errors in Xcode...
I don't understand why...
Thx
1 · Reply · Share ›

VvV ➚ Pakap84 • 4 years ago
I face the same issue. Did you figure out the problem.
Any help appreciated.

Thanks.
1 · Reply · Share ›

RLG02 ➚ VvV • 4 years ago
See comment above about this issue-- you may need to add the storyboard to your project. Or you may also need to comment out (or remove) the window creation and display code in AppDelegate.m. This is explained in the original instructions above.
· Reply · Share ›

jagveer Rana ➚ Pakap84 • 3 years ago
i have same problem ,, any body can help me??
· Reply · Share ›

manoj shivhare • 24 days ago
hey guys, i am facing problem..
Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '-[AppDelegate managedObjectContext]:
· Reply · Share ›

Kashyap Bhatt • 3 months ago
Hello,

I am getting this error:

Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+e
NSManagedObjectContext parameter searching for entity name 'Device''
· Reply · Share ›

Alexandru • 4 months ago
I already worked with CoreData before but some time passed since then. And it looks li

I included CoreData from the beginning, but I couldn't access the ManagedObjectCont
the tutorial. So this didn't work anymore. Looks like Apple changed something since th
the PersistentContainer.

```
- (NSManagedObjectContext *)managedObjecteContext{

NSManagedObjectContext *context = nil;
id delegate = (AppDelegate *)[UIApplication sharedApplication].delegate;
if( [delegate performSelector:@selector(persistentContainer)] ){

context = [[delegate persistentContainer] viewContext];
}
return context;
}
```

Like in Swift, try to use the (AppDelegate *) class specification. The swift pendant is the

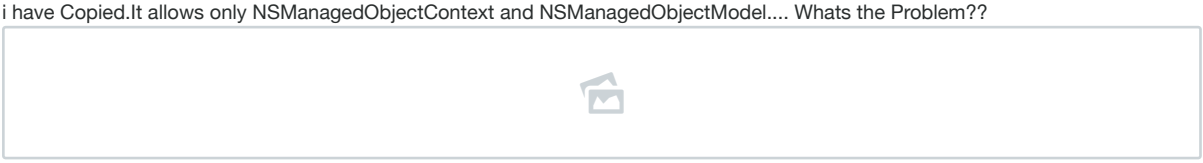
```
let context = (UIApplication.shared().delegate as! AppDelegate
```
· Reply · Share ›

Hishou • 6 months ago
Would it be possible to update the Core Data series of tutorials for Swift 3? thanks!
· Reply · Share ›

kuldeep tanwar • 8 months ago
how can you miss
@import CoreData;
Reply • Share ›

Napster • a year ago
Hi The DeviceDetail Page crashes as soon as I enter the input in textboxes. Any help will be much appreciated?
Reply • Share ›

Raghunath • a year ago
I am not able to create NSManagedObject instance (*newDevices).It doesn't allow me to write NSManagedObject.In Screenshot ,Just i have Copied.It allows only NSManagedObjectContext and NSManagedObjectModel.... Whats the Problem??



Reply • Share ›

Raghunath ➚ Raghunath • a year ago
provide me solution.....
Reply • Share ›

Guest ➚ Raghunath • a year ago
Try importing CoreData at the top.
#import <coredata coredata.h="">
Reply • Share ›

Guest • 2 years ago
view is not dismissing.....

- (IBAction)cancel:(id)sender {

[self dismissViewControllerAnimated:YES completion:NULL];

}
Reply • Share ›

www.imtikon.com • 2 years ago
this is a good one as a tutorial
Reply • Share ›

Czarek • 3 years ago
Add button does not trigger seque, anyone know how to fix that?
Reply • Share ›

Load more comments

PREVIOUS POST
iOS Programming 101: How To Customize Tab Bar Background and Appearance

NEXT POST
Core Data Part 2: Update, Delete Managed Objects and View Raw SQL Statement