Core Data

Introduction

Core Data is an API interface that helps create and persist the model layer of the iOS app. Core Data can seem intimidating at first but you'll soon see it's not that difficult. Taking the time to learn Core Data will help you save some time in the future. Also, learning the ups and downs will also give you an idea if it is the best choice for the app your building, maybe another persistent technology would better suite its needs like Sqlite3 if your app should work for both iOS and Android, since you'll only need to design one schema for the data.

Below are some important concepts and definitions in Core Data. After that, there's a project walkthrough which will demonstate the basics of using and setting up Core Data.

Managed Objects

These are object models that have a tie in to the Core Data Framework. These object models have properties and can also have relationships with other managed objects.

Managed Objects Model

Core Data is not a database, however, it does share a few common concepts with it. Representing objects is similar to how we represent them in traditional database systems. Instead of tables, we have entities, and instead of columns we have attributes.

Managed Object Context

Apple describes it in the following way:

> You can think of a managed object context as an intelligent scratch pad. When you fetch objects from a persistent store, you bring temporary copies onto the scratch pad where they form an object graph (or a collection of object graphs). You can then modify those objects however you like. Unless you actually save those changes, however, the persistent store remains unaltered.

Simply, it is a temporary point where your fetched objects, and the changes you make to them are stored, until you save them in the persistent store.

Presistent Store Coordinator

A communication link between the managed object context and the persistent store.

The Project

We will create a simple application that will demonstrate the use of Core Data. Our application is

simple version of the iPhone's contacts app that will store basic contact information as well as an image of the contact.

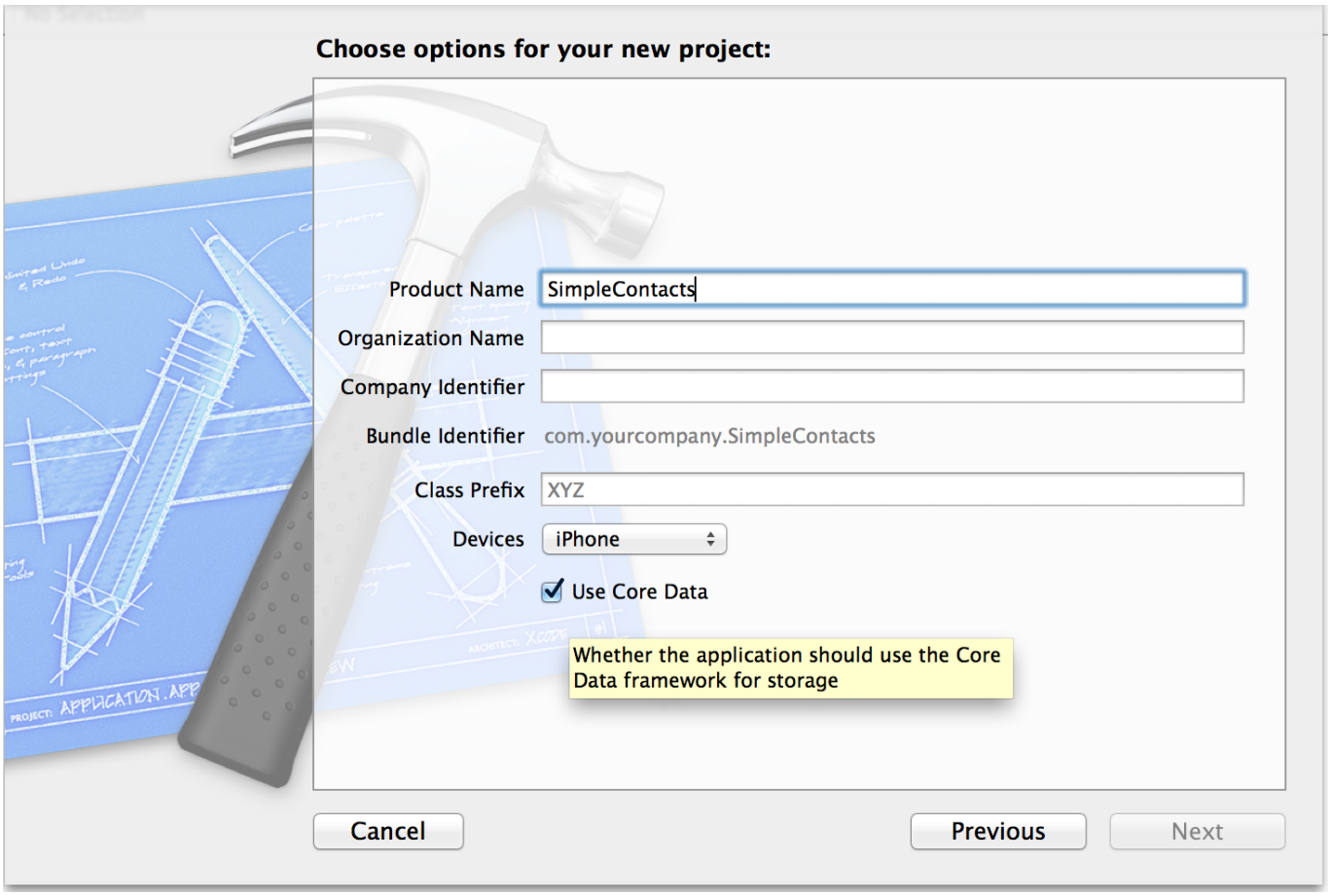
This project will demonstrate how to: * Set up CoreData * Setting up an Object Model * Saving objects. * Retrieving objects. * Updating objects.

Here's the link to the completed project that you can explore.

Creating the Project

Create a new empty project with the name "SimpleContacts" and be sure to check "Use Core Data".

If we look into our AppDelegate.m, we'll see that Xcode added all the necessary core data methods for us, which is convenient. But, the AppDelegate isn't the best place for them so we'll move them to a new class. If you forgot to check "Use Core Data", no problem, just add the CoreData.framwork to your project. And add Command+N -> Core Data -> Data Model and name it "SimpleContacts".



Creating the CoreDataStack

We need to clean up our app delegate before we start. Remove all the commented out code below from your AppDelegate.h:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
```

```
    //self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
    // Override point for customization after application launch.
    //self.window.backgroundColor = [UIColor whiteColor];
    //[self.window makeKeyAndVisible];
    return YES;
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    // Saves changes in the application's managed object context before the
application terminates.
    //    [self saveContext];
}
```

Now remove all three @synthesize from the implementation file, as well as their corresponding properties in the header file, but keep the UIWindow property. Remove the saveContext and applicationDocumentsDirectory methods declarations from the header as well. You'll notice that we have a lot of errors now, but we're going to fix them.

Create a new NSObject and name it CoreDataStack. Copy all the methods below the `applicationWillTerminate` to our CoreDataStack.m file and delete them from the AppDelegate.m.

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


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

+(instancetype) defaultStack;
```

Add the above code to your CoreDataStack.h file.

And then the following to CoreDataStack.m:

```
@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;



+(instancetype) defaultStack
{
```

```
    static CoreDataStack *defaultStack;

    //Guranteed to excute once
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        defaultStack = [[self alloc] init];
    });

    return defaultStack;
}
```

We just added the methods the properties we previously deleted from our AppDelegate and then we created a method that will return an instance of our Core Data Stack so we can use it elsewhere. This method also guarantees that only one instance of our Core Data Stack is available throughout our program's lifetime for efficiency reasons.

Your CoreDataStack.m should be similar to the one below:

```
#import "CoreDataStack.h"

@implementation CoreDataStack

@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;



+(instancetype) defaultStack
{
    static CoreDataStack *defaultStack;

    //Guranteed to excute once
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        defaultStack = [[self alloc] init];
    });

    return defaultStack;
}

- (void)saveContext
{
    NSError *error = nil;
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil) {
        if ([managedObjectContext hasChanges] && ![managedObjectContext
save:&error]) {
            // Replace this implementation with code to handle the error
appropriately.
            // abort() causes the application to generate a crash log and terminate.
You should not use this function in a shipping application, although it may be
```

```
useful during development.
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}

#pragma mark - Core Data stack

// Returns the managed object context for the application.
// If the context doesn't already exist, it is created and bound to the persistent
store coordinator for the application.
- (NSManagedObjectContext *)managedObjectContext
{
    if (_managedObjectContext != nil) {
        return _managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (coordinator != nil) {
        _managedObjectContext = [[NSManagedObjectContext alloc] init];
        [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    }
    return _managedObjectContext;
}

// 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:@"SimpleContacts"
withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

// Returns the persistent store coordinator for the application.
// If the coordinator doesn't already exist, it is created and the application's
store added to it.
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator
{
    if (_persistentStoreCoordinator != nil) {
        return _persistentStoreCoordinator;
    }

    NSURL *storeURL = [[self applicationDocumentsDirectory]
URLByAppendingPathComponent:@"SimpleContacts.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.

         abort() causes the application to generate a crash log and terminate. You
should not use this function in a shipping application, although it may be useful
during development.

         Typical reasons for an error here include:
         * The persistent store is not accessible;
         * The schema for the persistent store is incompatible with current managed
object model.
         Check the error message to determine what the actual problem was.


         If the persistent store is not accessible, there is typically something
wrong with the file path. Often, a file URL is pointing into the application's
resources directory instead of a writeable directory.

         If you encounter schema incompatibility errors during development, you can
reduce their frequency by:
         * Simply deleting the existing store:
         [[NSFileManager defaultManager] removeItemAtURL:storeURL error:nil]

         * Performing automatic lightweight migration by passing the following
dictionary as the options parameter:
         @{NSMigratePersistentStoresAutomaticallyOption:@YES,
NSInferMappingModelAutomaticallyOption:@YES}

         Lightweight migration will only work for a limited set of schema changes;
consult "Core Data Model Versioning and Data Migration Programming Guide" for
details.

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

    return _persistentStoreCoordinator;
}

#pragma mark - Application's Documents directory

// Returns the URL to the application's Documents directory.
- (NSURL *)applicationDocumentsDirectory
{
    NSURL *url = [[[NSFileManager defaultManager]
URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] lastObject];

    NSLog(@"Database URL: %@", url);
    return url;
}

@end
```

Creating our Interface

We need to add a storyboard to our project so press Command+N -> User Interface -> Storyboard then click next. Set the device family to iPhone and call it MainStoryBoard.

Now we need to inform our project to use it as our main interface. So go to the the projects General settings then Deployment Info -> Main Interface and choose MainStoryboard.

Add a Table View Controller to our storyboard and embed it in a navigation controller (Click the controller then Editor->Embed In-> Navigation Controller).

The app should run now and you'll be presented with an empty tableView.

Add a Bar Button Item on the table view's navigation bar, and in the attributes pane set the Identifier to Add.

Add a View Controller, add two bar buttons to it, on the left is cancel and on the right is save.

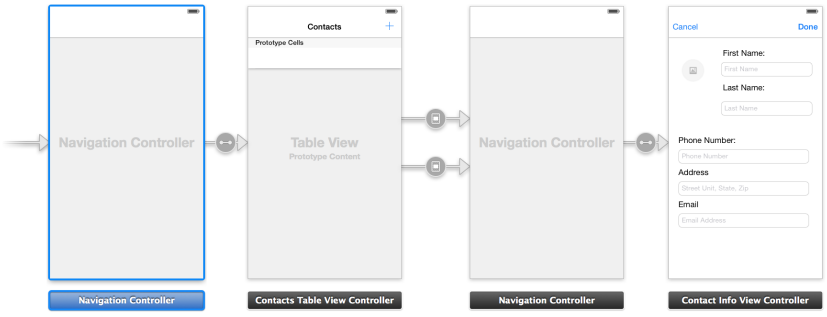
Then add five labels and textfields for the following properties: * First Name * Last Name * Phone Number * Address * Email

Also add a button to the right of these fields.

Download this image and add it to your project, then set it as the image to your button.

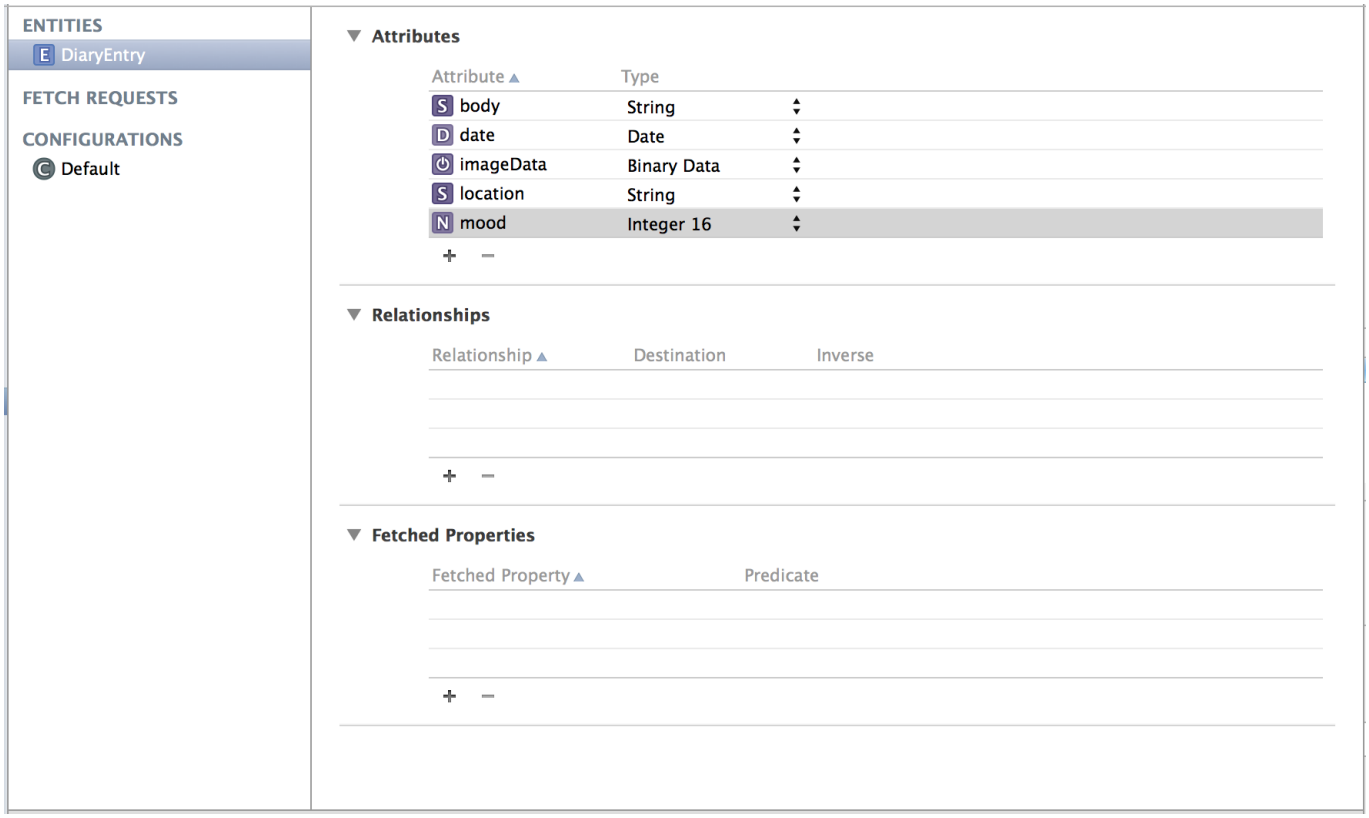
Link:

Try to make your Storyboard look like this:



Creating the Object Model

Now go to SimpleContacts.xcdatamodeld, click add entity and name it "ContactEntity" and then add the attributes as in the image below by clicking on Add Attribute which should be at the bottom of the screen.



Core Data doesn't save images directly, it doesn't have an image type, but luckily we can store the images as normal binary data using NSData, you'll see how that is done in the "Adding Images" section.

Now we need to create an NSManagedObject that models the entity we created in our object model.

Click Command+N -> Cocoa Touch-> Objectiec-C class. Name it ContactEntity and make it a subclass of NSManaged Object.

Add the following code to ContactEntity.h:

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

@interface ContactEntity : NSManagedObject


//Properties defined in our model
@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, strong) NSString *lastName;
@property (nonatomic, strong) NSString *nameFirstLetter;
@property (nonatomic, strong) NSString *email;
@property (nonatomic, strong) NSString *address;
//For simplicity we made the phoneNumber a string.
@property (nonatomic, strong) NSString *phoneNumber;
```

```
@property (nonatomic, retain) NSData * image;

@end
```

And the following to ContactEntity.m:

```
@implementation ContactEntity

//Because getters and setters are set in runtime
@dynamic firstName;
@dynamic nameFirstLetter;
@dynamic lastName;
@dynamic image;
@dynamic email;
@dynamic address;
@dynamic phoneNumber;


@end
```

Adding New Contacts

Create a new class called "ContactInfoViewController" and make it a subclass of UIViewController. Next, link up all the textfields and buttons from the view controller on our storyboard to our implementation file below the @interface.

Give them the same names as in the code below:

```
@interface ContactInfoViewController ()


//TextFields
@property (strong, nonatomic) IBOutlet UITextField *firstNameTextField;
@property (strong, nonatomic) IBOutlet UITextField *lastNameTextField;
@property (strong, nonatomic) IBOutlet UITextField *phoneNumberTextField;
@property (strong, nonatomic) IBOutlet UITextField *addressTextField;
@property (strong, nonatomic) IBOutlet UITextField *emailTextField;

//Buttons
@property (strong, nonatomic) IBOutlet UIButton *contactImageButton;

//Actions
- (IBAction)pressedCancel:(id)sender;
- (IBAction)pressedDone:(id)sender;


//Other
@property (nonatomic, strong) UIImage *pickedImage;
```

```
- (IBAction)pressedContactImageButton:(id)sender;
```

Now we need to import ContactEntity and the CoreDataStack to the implementation file.

Since we are creating a new contact and not retrieving it from Core Data, we don't need to set up an NSFetchRequest but we do need an instance of our CoreDataStack to save it.

Add `@property (nonatomic, strong) ContactEntity *contact;` to ContactInfoViewController.h and `@class ContactEntity;` before the @interface. We didn't import ContactEntity here because it would create a cycle, because it would cause make our app buggy to be buggy and most likely crash.

Add the following two methods to your implementation file:

```
- (IBAction)pressedDone:(id)sender
{
        [self addContact];
}



-(void) addContact
{
    if([self.firstNameTextField.text length] > 0)
    {
        //Adding an entry to our database
        CoreDataStack *coreDataStack = [CoreDataStack defaultStack];


        //Create a new object to save in coreData; must have the same name as the
enity in the model.
        //Same as inserting a new row in a database.
        ContactEntity *contact = [NSEntityDescription
insertNewObjectForEntityForName:@"ContactEntity"
inManagedObjectContext:coreDataStack.managedObjectContext];

        [self setContactDataFromViewData:contact];


        //Save the new object
        [coreDataStack saveContext];

        [self dismiss];
    }
    else
    {


        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Can't Create
Contact!!" message:@"A contact must have a first name."  delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil];
```

```
        [alert show];
    }
}

-(void) setContactDataFromViewData: (ContactEntity *) contact
{

        NSString *firstName = self.firstNameTextField.text;
        contact.firstName = firstName;

        contact.lastName = self.lastNameTextField.text;
        contact.phoneNumber = self.phoneNumberTextField.text;
        contact.address = self.addressTextField.text;
        contact.email = self.emailTextField.text;

        unichar firstLetter = [firstName characterAtIndex:0] ;
        contact.nameFirstLetter = [NSString stringWithCharacters:&firstLetter
length:1];

        NSLog(@"Char is %@", contact.nameFirstLetter);


        if(self.pickedImage != nil)
            contact.image = UIImageJPEGRepresentation(self.pickedImage, 0.75);

}
```

That should be enough to create a contact without a picture. Now go back to the storyboard, control drag from the add button in the tableViewController to the viewController and don't forget to set the customClass in the identity pane to "ContactInfoViewController".

Displaying the Contacts on the Table

Create a new class called "ContactInfoViewController" and make it a subclass of UIViewController.

Set the TableViewController's custom class to ContactInfoViewController in the story board. Also, set the cellIdentifier in the storyboard to "Cell".

```
#pragma mark - Helper Methods


-(NSFetchRequest *) entryListFetchRequest
{
    //Fetch our contacts
    NSFetchRequest *fetchRequest = [NSFetchRequest
fetchRequestWithEntityName:@"ContactEntity"];

    //Sort descendingly based on firstName
    fetchRequest.sortDescriptors = @[[NSSortDescriptor
sortDescriptorWithKey:@"nameFirstLetter" ascending:YES]];

    return fetchRequest;
}
```

```
-(NSFetchedResultsController *) fetchedResultsController
{
    if(_fetchedResultsController != nil)
    {
        return _fetchedResultsController;
    }
    else
    {
        CoreDataStack *coreDataStack = [CoreDataStack defaultStack];

        NSFetchRequest *fetchRequest = [self entryListFetchRequest];

        _fetchedResultsController =  [[NSFetchedResultsController alloc]
                                      initWithFetchRequest:fetchRequest

managedObjectContext:coreDataStack.managedObjectContext
                                     sectionNameKeyPath:@"nameFirstLetter"
                                     cacheName:nil];


        _fetchedResultsController.delegate = self;

        return _fetchedResultsController;
    }
}
```

Those are the general methods to retrieve the data stored in our persistent Store.

This is what you should have in ContactsTableViewController.m:

```
#import "ContactsTableViewController.h"
#import "CoreDataStack.h"
#import "ContactEntity.h"
#import "ContactInfoViewController.h"

@interface ContactsTableViewController () <NSFetchedResultsControllerDelegate>

@property (nonatomic, strong) NSFetchedResultsController *fetchedResultsController;


@end

@implementation ContactsTableViewController


- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.fetchedResultsController performFetch:nil];

}
-(void)controllerDidChangeContent:(NSFetchedResultsController *)controller
{
```

```
    [self.tableView reloadData];
}

#pragma mark - Table view data source

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



-(NSString *) tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
    id<NSFetchedResultsSectionInfo> sectionInfo = [self.fetchedResultsController
sections][section];

    return [sectionInfo indexTitle];
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{

    id<NSFetchedResultsSectionInfo> sectionInfo = [self.fetchedResultsController
sections][section];
    // Return the number of rows in the section.
    return  [sectionInfo numberOfObjects];

}

//Enable Delete
-(UITableViewCellEditingStyle) tableView:(UITableView *)tableView
editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return UITableViewCellEditingStyleDelete;
}

//Delete the contact
-(void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath
{
    //Get the selected contact
    ContactEntity *contact = [self.fetchedResultsController
objectAtIndexPath:indexPath];
    CoreDataStack *coreDataStack = [CoreDataStack defaultStack];

    //Delete internally
    [[coreDataStack managedObjectContext] deleteObject:contact];

    [coreDataStack saveContext];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
```

```
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"
forIndexPath:indexPath];

    ContactEntity *contact = [self.fetchedResultsController
objectAtIndexPath:indexPath];

    cell.textLabel.text = [NSString stringWithFormat:@"%@ %@", contact.firstName,
contact.lastName];


    return cell;
}




#pragma mark - Helper Methods


-(NSFetchRequest *) entryListFetchRequest
{
    //Fetch our contacts
    NSFetchRequest *fetchRequest = [NSFetchRequest
fetchRequestWithEntityName:@"ContactEntity"];

    //Sort decendingly based on firstName
    fetchRequest.sortDescriptors = @[[NSSortDescriptor
sortDescriptorWithKey:@"nameFirstLetter" ascending:YES]];

    return fetchRequest;
}

-(NSFetchedResultsController *) fetchedResultsController
{
    if(_fetchedResultsController != nil)
    {
        return _fetchedResultsController;
    }
    else
    {
        CoreDataStack *coreDataStack = [CoreDataStack defaultStack];

        NSFetchRequest *fetchRequest = [self entryListFetchRequest];

        _fetchedResultsController =  [[NSFetchedResultsController alloc]
                                      initWithFetchRequest:fetchRequest

managedObjectContext:coreDataStack.managedObjectContext
                                     sectionNameKeyPath:@"nameFirstLetter"
                                     cacheName:nil];


        _fetchedResultsController.delegate = self;

        return _fetchedResultsController;
```

```
    }
}
```

This code allows you to delete contacts by swiping to the left. And, it also divides our contacts into sections based on the first letter of their first name. We added the firstLetter property to our model because that is the easiest way to do it.

Editing Contacts

First add a modal segue from the ContactsTableViewController to ContactInfoViewController, and give it the identifier "Edit".

Add the following to ContactsTableViewController.m:

```
#pragma mark - Navigation

-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if([segue.identifier isEqual:@"Edit"])
    {
        UITableViewCell *cell = sender;
        NSIndexPath *indexPath = [self.tableView indexPathForCell:cell];

        UINavigationController *navigaitonController = [segue
destinationViewController];

        ContactInfoViewController *contactInfoViewcontroller =
(ContactInfoViewController *)navigaitonController.topViewController;

        contactInfoViewcontroller.contact = [self.fetchedResultsController
objectAtIndexPath:indexPath];
    }
}
```

Go back to ContactInfoViewController.m. Update the viewDidLoad method and the pressedDone method to:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    //If editing contact info
    if(self.contact != nil)
    {
        [self setViewDataFromContactData];


    }
```

```
}

- (IBAction)pressedDone:(id)sender
{

    if (self.contact != nil)
    {
        [self updateContact];

    }
    else
    {
        [self addContact];
    }


}
```

Add the following:

```
-(void) setViewDataFromContactData
{
    self.firstNameTextField.text = self.contact.firstName;
    self.lastNameTextField.text = self.contact.lastName;
    self.phoneNumberTextField.text = self.contact.phoneNumber;
    self.addressTextField.text = self.contact.address;
    self.emailTextField.text = self.contact.email;

}


-(void) updateContact
{

    //There might be better ways to doing this, but this is just a basic demo
    [self setContactDataFromViewData:self.contact];


    CoreDataStack *coreDataStack = [CoreDataStack defaultStack];
    [coreDataStack saveContext];

    [self dismiss];

}
```

Oh, I forgot to mention that we should also add the UINavigationControllerDelegate to our interface `@interface ContactInfoViewController () <UIActionSheetDelegate>`.

That's it for updating.

Adding images

We're going to use standard methods that give the user an option to pick a picture from the photoroll or, take a new picture, and then save the selected image to our project.

Add the following to ContactInfoViewController.m:

```
- (IBAction)pressedContactImageButton:(id)sender
{
    if([UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera])
    {
        [self promptForSource];
    }
    else
    {
        [self promptForPhotoRoll];
    }
}


#pragma mark - Image Picker Methods
-(void)promptForSource
{
    UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:@"Image
Source" delegate:self cancelButtonTitle:@"Cancel" destructiveButtonTitle:nil
otherButtonTitles:@"Camera", @"PhotoRoll", nil];

    [actionSheet showInView:self.view];

}


-(void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex != actionSheet.cancelButtonIndex)
    {
        if (buttonIndex != actionSheet.firstOtherButtonIndex) {
            [self promptForPhotoRoll];
        }
        else
        {
            [self promptForCamera];
        }
    }
}

-(void)promptForCamera
{
    UIImagePickerController *controller = [[UIImagePickerController alloc] init];
    controller.sourceType = UIImagePickerControllerSourceTypeCamera;
    controller.delegate = self;
    [self presentViewController:controller animated:YES completion:nil];
```

```
}

-(void)promptForPhotoRoll
{
    UIImagePickerController *controller = [[UIImagePickerController alloc] init];
    controller.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    controller.delegate = self;
    [self presentViewController:controller animated:YES completion:nil];
}

-(void) imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    UIImage *image =  info[UIImagePickerControllerOriginalImage];
    self.pickedImage = image;

    [self dismissViewControllerAnimated:YES completion:nil];
}

-(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [self dismissViewControllerAnimated:YES completion:nil];
}

-(void) setPickedImage:(UIImage *)pickedImage
{
    _pickedImage = pickedImage;
    if (pickedImage == nil) {
        [self.contactImageButton setImage:[UIImage imageNamed:@"icn_noimage"]
forState:UIControlStateNormal];

    }
    else
    {
        [self.contactImageButton setImage:pickedImage
forState:UIControlStateNormal];
    }
}
```

Update viewDidLoad to round off the corners of the image by appending

```
    //Round the corners of our image button
    self.contactImageButton.layer.cornerRadius =
CGRectGetWidth(self.contactImageButton.frame)/2.0f;
```

to the end of theViewDidLoad method.

Update the interface with the following delegates:

```
@interface ContactInfoViewController () <UIActionSheetDelegate,
UINavigationControllerDelegate,
UIImagePickerControllerDelegate>
```

Go back to ContactsTableViewController.m update the cellForRowAtIndexPath method with this line `cell.imageView.image = [UIImage imageWithData:contact.image];`.

That is it for the project.

Summary

Setting up CoreData:

There are two ways to set up coreData, either create the project from the beginning with CoreData, and move the CoreDataStack methods and properties from the AppDelegate to a separate method. (You can skip this step if you prefer keeping them in the AppDelegate)

Or, if you already began working on your project and you realized that you need or want to use CoreData to store information. Follow the steps below:

1. Add CoreData.framework to your project.
2. Download CoreDataStack.h and CoreDataStack.m and add them to your project.
3. Command+N -> Core Data -> Data Model and give it a name. It is preferable if it has the same name as your project.
4. Change the modelURL in the managedObjectModel with the name of your object model. Look at the example.
5. Change the storeURL property in the persistentStoreCoordinator method with the name of your object model. Look at the example.

Example: Your object model has the name "testProject"

Original:

```
// 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:@"SimpleContacts"
withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

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

```
    NSURL *storeURL = [[self applicationDocumentsDirectory]
URLByAppendingPathComponent:@"SimpleContacts.sqlite"];
    .
    .
    .
    .
    .
    .
}
```

Changed:

```
// 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:@"testProject"
withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

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

    NSURL *storeURL = [[self applicationDocumentsDirectory]
URLByAppendingPathComponent:@"testProject.sqlite"];
    .
    .
    .
    .
    .
    .
}
```

In order to interact with the your data model you'll need to create an NSManaged object with the same properties as in your model. Also, instead of using `@synthesize` use `@dynamic`. You'll also need need to import CoreData into your project , just as in the SimpleContacts project.

Adding Objects:

1. Create and initialize an instance of your NSManagedObject by using and modifying the following:

```
```

```
[NSEntityDescription
insertNewObjectForEntityForName:@"nameOfYourEntityAsDefinedInTheObjectModel"
 inManagedObjectContext:coreDataStack.managedObjectContext];```
2. Import ```CoreDataStack.h``` and create a CoreDataStack with the default stack.
3. Set the values for your managed Object, this will save it in the Managed Objects
Context.
4. When you want to officially save it (pressit it to the store) use the default
stack and
call saveContext.```[coreDataStack saveContext];```

###Updating Objects

1. Have a property with the desired object.
2. Change its values.
3. Import ```CoreDataStack.h``` and create a CoreDataStack with the default stack.
4. Save the context using your default CoreDataStack.

###Retrieving Objects

1. Add ```NSFetchedResultsControllerDelegate``` to your objects interface.
2. Add the general NSFetchRequest methods.
3. Call ``` [self.fetchedResultsController performFetch:nil]; ``` in your
viewDidLoad method.
4. To retrieve an object at a detain index path call ```
[self.fetchedResultsController objectAtIndexPath:indexPath];
 ```.


Genreal NSFetch Request Methods:
```

-(NSFetchRequest *) entryListFetchRequest { //Fetch the entities NSFetchRequest *fetchRequest = [NSFetchRequest fetchRequestWithEntityName:@"NameOfYourEntityAsDefinedInTheModel"];

```
//Sort descendingly
fetchRequest.sortDescriptors = @[[NSSortDescriptor
sortDescriptorWithKey:@"thePropertyYouWantToOrderTheResultsBy" ascending:YES]];

return fetchRequest;
```

}

-(NSFetchedResultsController *) fetchedResultsController { if(_fetchedResultsController != nil) { return _fetchedResultsController; } else { CoreDataStack *coreDataStack = [CoreDataStack defaultStack];

```
    NSFetchRequest *fetchRequest = [self entryListFetchRequest];

    _fetchedResultsController =  [[NSFetchedResultsController alloc]
                                  initWithFetchRequest:fetchRequest

managedObjectContext:coreDataStack.managedObjectContext
```
```

```
    sectionNameKeyPath:@"thePropertyYouWantToUseToSectionOffYourResults"
                                    cacheName:nil];


    _fetchedResultsController.delegate = self;

    return _fetchedResultsController;
}
```

}

```

### Deleting an Object:

1. Have a property with the desired object.
2. Import `CoreDataStack.h` and create a CoreDataStack with the default stack.
3. Call `[[coreDataStack managedObjectContext] deleteObject:object];` with the object you want to delete.
4. Save the context using your default CoreDataStack.

## Additional Links:

- Simple Contacts Download.
- Apple's Core Data Programming Guide.
- Raywenderlick iOS5 Core Data Tutorial part1, part2, and part3
- A full Core Data application fron the objc periodical.
```