iOS Core Data Example Application

The Core Data framework provides an abstract, object oriented interface to database storage within iOS applications. This does not require extensive knowledge of database or the use of files to manage data.

Step 1: Create an **Empty Application** in Xcode. Currently, only the Master-Detail Application, Utility Application and Empty Application project templates offer the option to automatically include support for Core Data.

To create the example application project, launch Xcode and select the option to create a new project. In the new project window, select the **Empty Application option**. In the next screen make sure that the Devices menu is set to iPhone and that the check boxes next to **Use Core Data** and **Use Automatic Reference Counting** are selected. Save the project as "CoreData" or anything else that you like.

In addition to the usual files that are present when creating a new project, this time an additional file named **CoreData.xcdatamodeld** is also created. This is the file where the entity descriptions for your data model is going to be stored.

Step 2: Creating the Entity Description

The entity description defines the model for our data, much in the way a schema defines the model of a database table. To create the entity for the Core Data application, select the CoreData.xcdatamodeld file to load the entity editor:

To create a new entity, click on the **Add Entity button** located in the bottom panel. In the text field that appears beneath the Entities heading name the entity **Contacts**.

With the entity created, the next step is to add some attributes that represent the data that is to be stored. To do so, click on the **Add Attribute button**. In the Attribute pane, name the attribute **name** and set the Type to **String**. Repeat these steps to add two other String attributes **address** and **phone** respectively. The finished set of attributes looks like this:

The entity is now defined.

Step 3: Adding a View Controller

In order to automatically add Core Data support to our application we had to choose the Empty Application project template option when we started Xcode. As such, we now need to create our own view controller.

Within the Xcode project navigator panel, Ctrl-click on the CoreData folder entry. From the popup menu, select *New File...*. In the new file panel, select the iOS Cocoa Touch Class category followed by the Objective-C class icon and click Next. On the following options screen, make sure the Subclass of menu is set to **UIViewController** and name the class **CoreDataViewController**. Select the **With XIB** for user interface check box and make sure the **Targeted for iPad option is off**. Click "next" and on the final panel click on "create."

Step 4: Modify app delegate to use the new controller. Now that we have added the view controller class to the application we need to **modify our app delegate to make this the root view controller**. In the Xcode project navigator, select the **AppDelegate.h** file and modify it to add a reference to our new view controller:

At the top of the file add:

```
#import "CoreDataViewController.h"

@class CoreDataViewController;
```

Between @interface and @end add:

```
@property (strong, nonatomic) CoreDataViewController
*viewController;
```

With an instance of the view controller declared in the interface file we now need to modify the **didFinishLaunchingWithOptions** method located in the **AppDelegate.m** implementation file to initialize and allocate the CoreDataViewController instance and assign it as the application's root view controller so that it is visible to the user on application launch:

Add the following lines to the bottom of the "**application**" method just before the "return YES;" line.

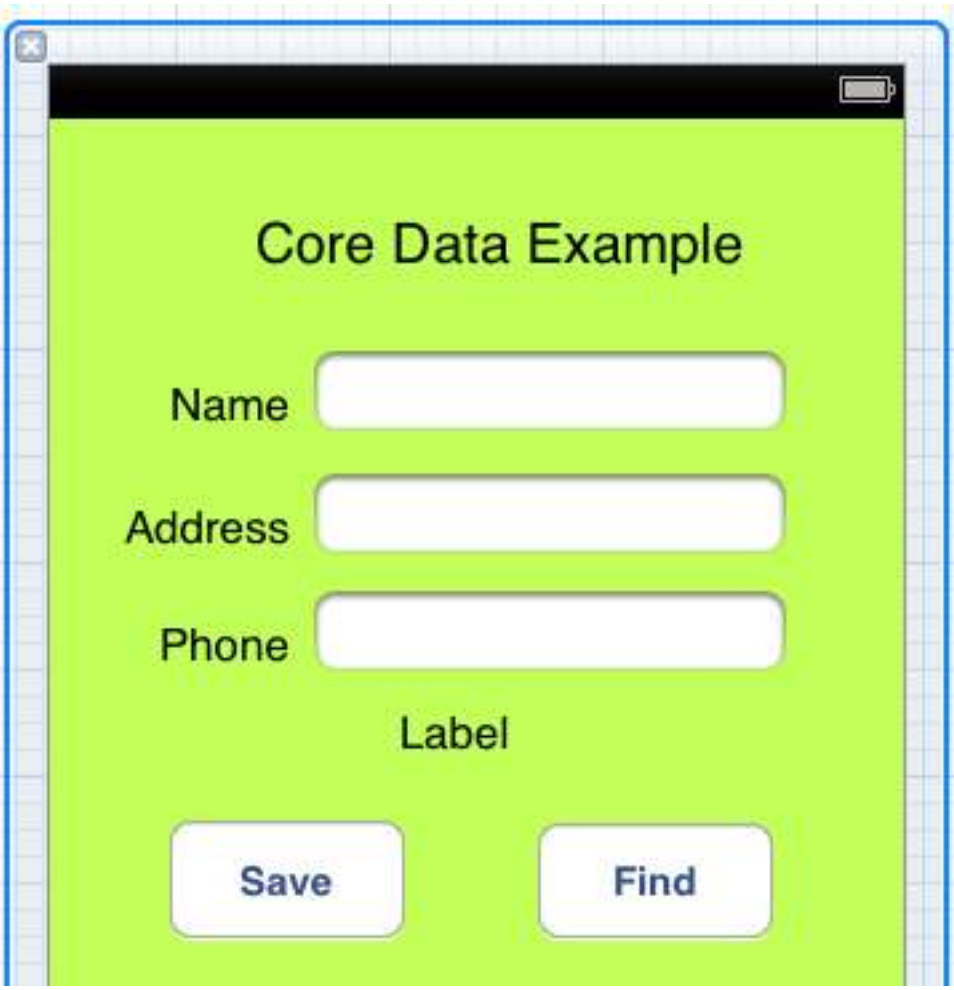
```
_viewController = [[CoreDataViewController alloc]
    initWithNibName:@"CoreDataViewController" bundle:nil];
    [self.window setRootViewController:_viewController];
```

Now your **CoreDataViewController.xib** interface will launch automatically when your application loads. Run your program to test it out and make sure all is loading correctly.

Step 5: Designing the User Interface

With the application delegate configured, now is a good time to design the user interface and establish the outlet and action connections.

Select the **CoreDataViewController.xib** file to begin the design work. The completed view should look like the following:



The user interface for an iOS Core Data Application example

Note: Before proceeding, stretch the status label (located above the two buttons) so that it covers most of the width of the view. Finally, edit the label and remove the word "Label" so that it is blank.

Step 6: Next you will wire the components to ViewController.h.

Select the top most text field object in the view canvas, display the Assistant Editor panel (tuxedo icon) and verify that the editor is displaying the contents of the ViewController.h file.

- Ctrl-click on the text field object and drag it to a position just below the @interface line. Release the line and in the resulting connection dialog establish an outlet connection named **theName**.

- Repeat the above steps to establish outlet connections for the remaining text fields and the label object to properties named **theAddress**, **thePhone** and **theStatus**.

- Ctrl-click on the Save button object and drag the line to the area immediately beneath the newly created outlet in the Assistant Editor panel. Release the line and, within the resulting connection dialog, establish an Action method on the Touch Up Inside or Touch Down event configured to call a method named **saveData**. Repeat this step to create an action connection from the Find button to a method named **findContact**.

On completion of these steps, the ViewController.h file should read as follows:

```
@property (weak, nonatomic) IBOutlet UITextField *theName;
@property (weak, nonatomic) IBOutlet UITextField *theAddress;
@property (weak, nonatomic) IBOutlet UITextField *thePhone;
@property (weak, nonatomic) IBOutlet UILabel *theStatus;

- (IBAction)saveData:(UIButton *)sender;
- (IBAction)findContact:(UIButton *)sender;
```

DON'T synthesize the properties inside of ViewController.m. Instead we are going to take advantage of an auto-synthesize feature of the latest iOS SDK. We get auto synthesize if we use the names above plus the underscore "_" in front of the name. For example:

theName is synthesized as **_theName**
theAddress is synthesized as **_theAddress**
thePhone is synthesized as **_thePhone**
theStatus is synthesized as **_theStatus**

Step 7: In the ViewController.h file, import the AppDelegate.h file which will be required by code added to the view controller later in this tutorial. Put the following underneath #import <UIKit/UIKit.h>:

```
#import "AppDelegate.h"
```

Step 8: Saving Data to the Persistent Store using Core Data. When the user touches the Save button the **saveData** method is called. It is within this method that we must implement the code to obtain the

managed object context and create and store managed objects containing the data entered by the user. Select the **ViewController.m** file, scroll down to the **saveData** method and implement the code as follows:

```
- (IBAction)saveData:(UIButton *)sender {
    AppDelegate *appDelegate =
    [[UIApplication sharedApplication] delegate];

    NSManagedObjectContext *context =
    [appDelegate managedObjectContext];
    NSManagedObject *newContact;
    newContact = [NSEntityDescription
                  insertNewObjectForEntityForName:@"Contacts"
                  inManagedObjectContext:context];
    [newContact setValue: _theName.text forKey:@"name"];
    [newContact setValue: _theAddress.text forKey:@"address"];
    [newContact setValue: _thePhone.text forKey:@"phone"];
    _theName.text = @"";
    _theAddress.text = @"";
    _thePhone.text = @"";
    NSError *error;
    [context save:&error];
    _theStatus.text = @"Contact saved";
}
```

The above code identifies the application delegate instance and uses that object to identify the managed object context. This context is then used to create a new managed object using the Contacts entity description. The setValue method of the managed object is then called to set the name, address and phone attribute values of the managed object (which in turn are read from the text field user interface components). Finally, the context is instructed to save the changes to the persistent store with a call to the context's save method.

Step 6: Retrieving Data from the Persistent Store using Core Data

In order to allow the user to search for a contact it is now necessary to implement the **findContact** action method. As with the save method, this method will need to identify the application delegate and managed object context. It will then need to obtain the entity description for the Contacts entity and then create a predicate to ensure that only objects

with the name specified by the user are retrieved from the store. Matching objects are placed in an array from which the attributes for the first match are retrieved using the **valueForKey** method and displayed to the user. A full count of the matches is displayed in the status field.

The code to perform these tasks is as follows:

```
- (IBAction)findContact:(UIButton *)sender {
    AppDelegate *appDelegate = [[UIApplication sharedApplication]
delegate];
    NSManagedObjectContext *context = [appDelegate
managedObjectContext];
    NSEntityDescription *entityDesc = [NSEntityDescription
entityForName:@"Contacts" inManagedObjectContext:context];
    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    [request setEntity:entityDesc];

    NSPredicate *pred = [NSPredicate predicateWithFormat:@"(name
= %@)", _theName.text];
    [request setPredicate:pred];
    NSManagedObject *matches = nil;

    NSError *error;
    NSArray *objects = [context executeFetchRequest:request
                                              error:&error];

    if ([objects count] == 0) {
        _theStatus.text = @"No matches";
    } else {
        matches = objects[0];
        _theAddress.text = [matches valueForKey:@"address"];
        _thePhone.text = [matches valueForKey:@"phone"];
        _theStatus.text = [NSString stringWithFormat: @"%d matches
found", [objects count]];
    }
}
```

Step 8. Optional: Make the keyboard go away when user finishes entering text. Otherwise, the keyboard might cover up the buttons to save and search.

Wire each of the textField, “Did End on Exit” events to your ViewController.h file as follows:

```
- (IBAction)nameDone:(UITextField *)sender;

- (IBAction)addressDone:(UITextField *)sender;

- (IBAction)phoneDone:(UITextField *)sender;
```

Add the following code to resign the first responded in your ViewController.m implementation of these methods:

```
- (IBAction)nameDone:(UITextField *)sender {
    [_theName resignFirstResponder];
}

- (IBAction)addressDone:(UITextField *)sender {
    [_theAddress resignFirstResponder];
}

- (IBAction)phoneDone:(UITextField *)sender {
    [_thePhone resignFirstResponder];
}
```

Step 8: Build and Run your application

Once the application compiles it will launch and load into the iOS Simulator. Enter some test contacts (some with the same name). Having entered some test data, enter the name of the contact for which you created duplicate records and click the Find button.