Session 5

Designing the iPhone User Interface

MODULE OBJECTIVE

At the end of this module, you will be able to:

- Build an interactive user interface for an app, where you can save and retrieve data

SESSION OBJECTIVES

At the end of this session, you will be able to:

- Introduce the Bands model object
- Build an interactive user interface
- Save and retrieve data

INTRODUCTION

Now that you know about the Xcode tool environment and the Swift language, you will learn about how to design the iPhone user interface. This is important for setting the view or user interface of your application. You can do so by creating an interactive user interface. You will also learn how to save and retrieve data of an app in an effective manner and display it to the user to make changes in it.

INTRODUCING THE BANDS MODEL OBJECT

iOS applications use the **Model-View-Controller** (MVC) design pattern. In the **Bands** app, the MVC model represents a band. Eventually, you will have multiple bands represented by the MVC model. So the first step is to create a class that encapsulates all the properties of a band within the application.

The band object needs the following properties:

- **Name** — The name of the band
- **Notes** — Any notes the user would like to attach to the band
- **Rating** — How the user rates the band on a scale of 1–10
- **Touring Status** — Whether the band is currently touring or if it is disbanded
- **Have Seen** — Whether the user has seen the band in a concert

Applications play an important role in our lives. Each iOS application is created with a purpose and with unique features. For an effective app, both features and look are important. Now, apart from Objective-C, Swift provides the ability to build an expressive, light, and fast-running app. The utility of an application is based on available features.

Suppose you want to perform some tough mathematical calculations. For this, you will need a calculator. However, you may not be familiar with the various features available in a calculator. This can be a tricky situation. To make it easy for yourself, you can create an application for calculation and incorporate only the relevant features of a calculator that suit your needs. This is just one example of the advantages of creating an iOS application.

Creating the Bands Model Object

The `WBABand` class will represent the **Bands** model object. The name of the class follows Apple's naming convention by adding a three-letter prefix to the beginning of the class name. The prefix is a combination of the company name, such as **Wrox** and the **Bands** app name.

Steps for Creating the `WBABand` Class

1. In Xcode, open the **Bands** project. If the project does not exist, then create a project.
2. Select **File** ➪ **New** ➪ **File**, as shown in **Figure 1**.

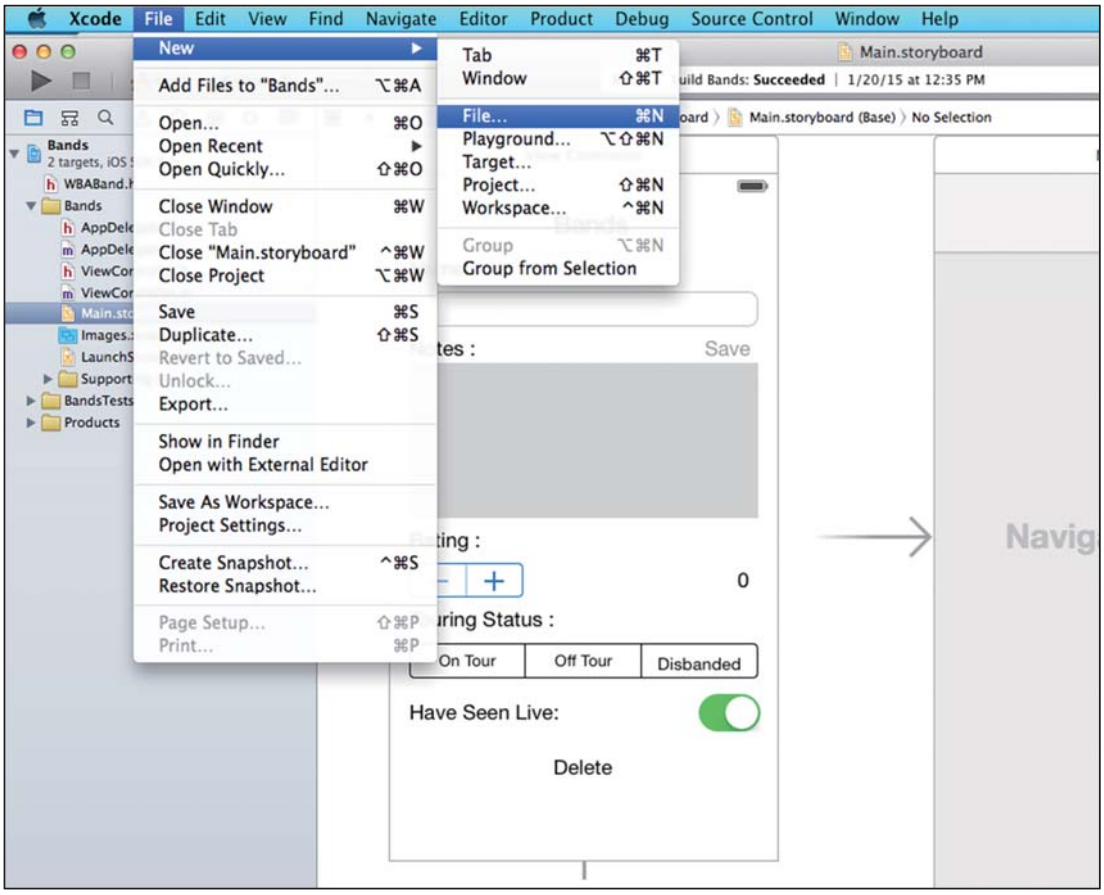

Figure 1: Create a New Project File

3. In the dialog box that opens, select the **Objective-C File** class and click **Next**, as shown in **Figure 2**.

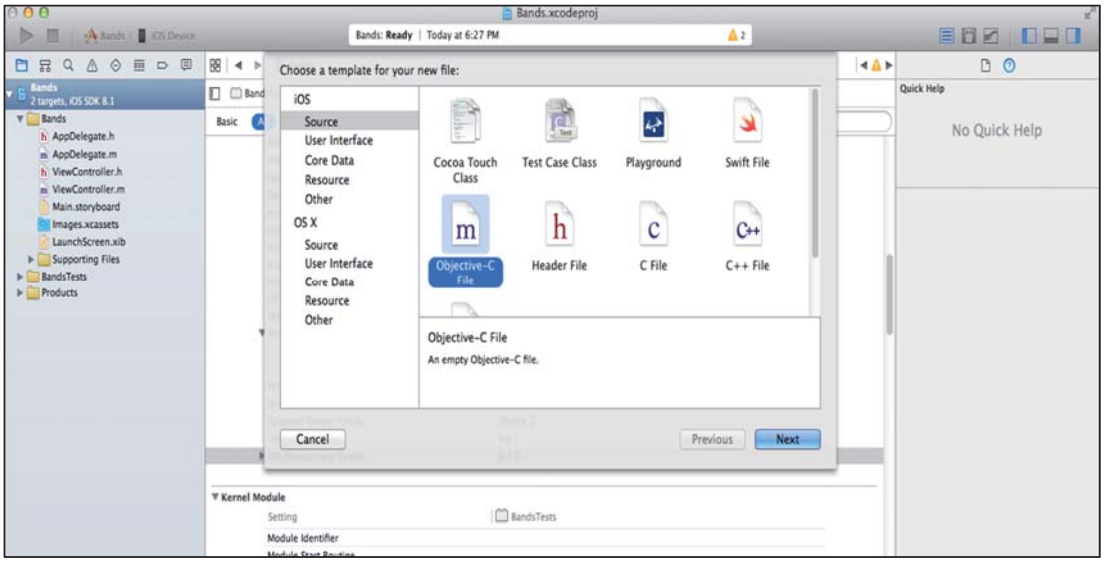

Figure 2: Select the Objective-C File Class

4. On the next page, name the class `WBABand`, set its subclass to `NSObject`, and click **Next**, as shown in **Figure 3**.

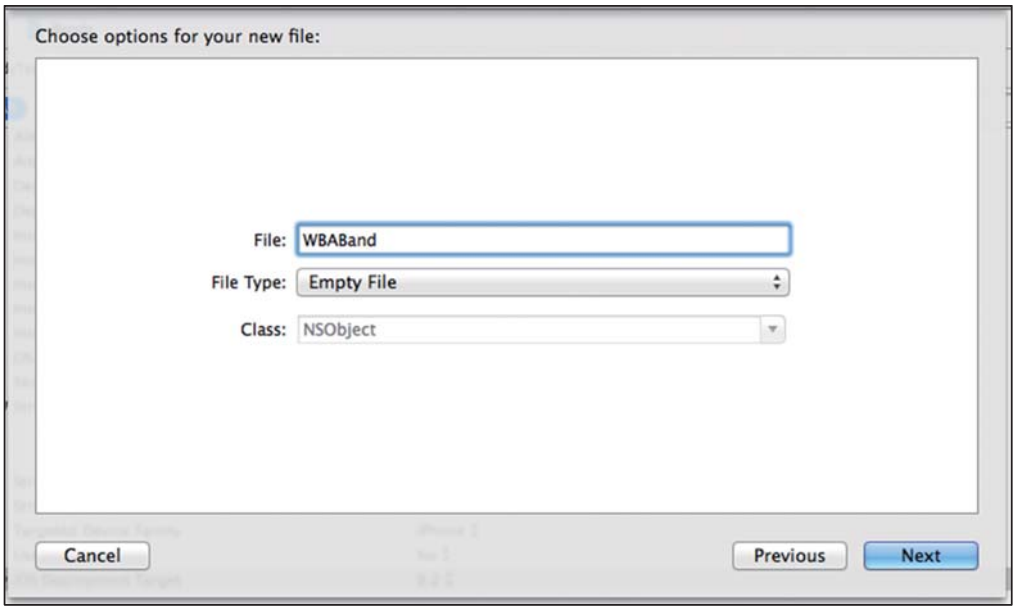

Figure 3: Select Options for the `WBABand` Class

5. The current file and rest of the project files will be saved into the **Bands** target folder, as shown in **Figure 4**.

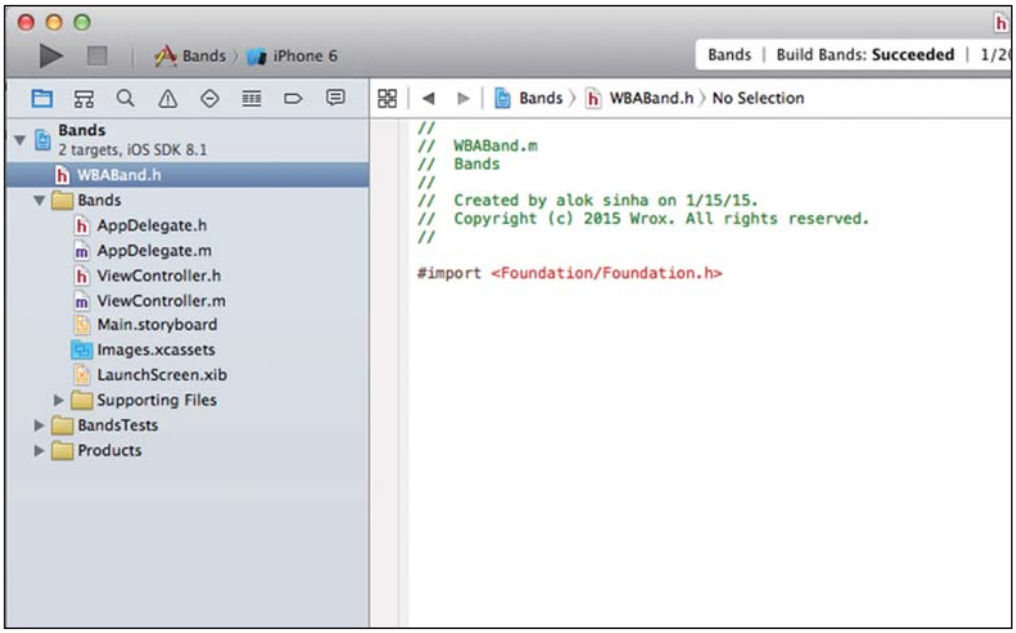

Figure 4: Save Files in the Bands Target folder

Creating Enumerations

Enumerations are common in most programming languages. They enable you to declare a type, which consists of named elements. The elements represent simple integers, but enable you to use their names in the code to make it readable.

Therefore, before adding properties to the WBABand class, you need to declare an enumeration to represent the following three different states of touring that a band can have:

- Touring
- Not Touring
- Disbanded

Touring, **Not Touring**, and **Disbanded** are the three states of a band. These are the variables created by an enumerator that will hold a value found only in an enumerator. Enumerators are declared with the help of the enum keyword, as follows:

- **Touring State:** The WBATouringStatusOnTour variable is used, which has the value 0.
- **Not Touring State:** The WBATouringStatusOffTour variable is used with the value 1.
- **Disbanded State:** The WBATouringStatusDisbanded variable is used with the value 2.

Steps for Creating an Enumeration

1. In Xcode, select the WBABand.h file in the Project Navigator, as shown in **Figure 5**.

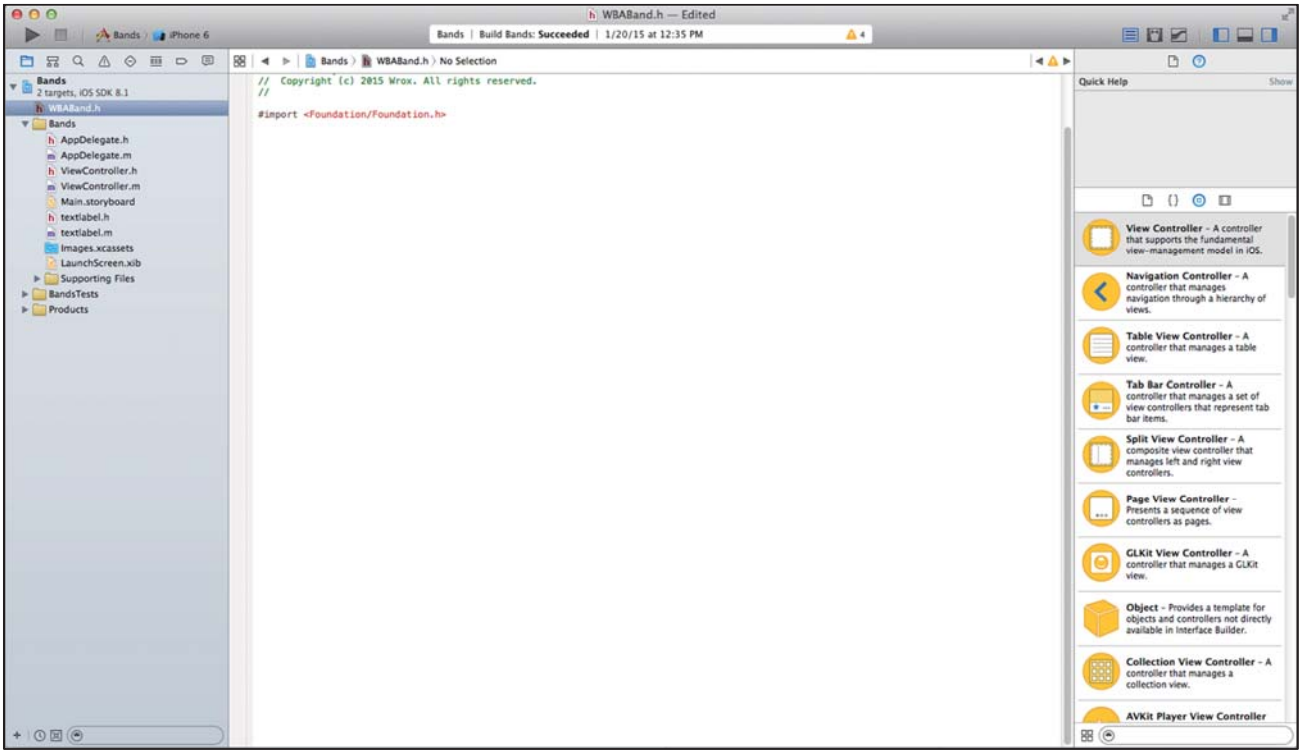

Figure 5: Select WBABand.h in Project Navigator

2. In the Code Editor, add the following code to the top of the file after the Import section, as shown in **Figure 6**:

```
typedef enum {
WBATouringStatusOnTour,
WBATouringStatusOffTour,
WBATouringStatusDisbanded,
} WBATouringStatus;
```

The syntax for declaring Touring state in Swift by the `enum` keyword is given below. New line members (`WBATouringStatusOnTour`, `WBATouringStatusOffTour`, `WBATouringStatusDisbanded`) in `enum` are defined using the `case` keyword.

```
// Swift Syntax
enum WBATouringStatus {
case WBATouringStatusOnTour
case WBATouringStatusOffTour
case WBATouringStatusDisbanded

}
```

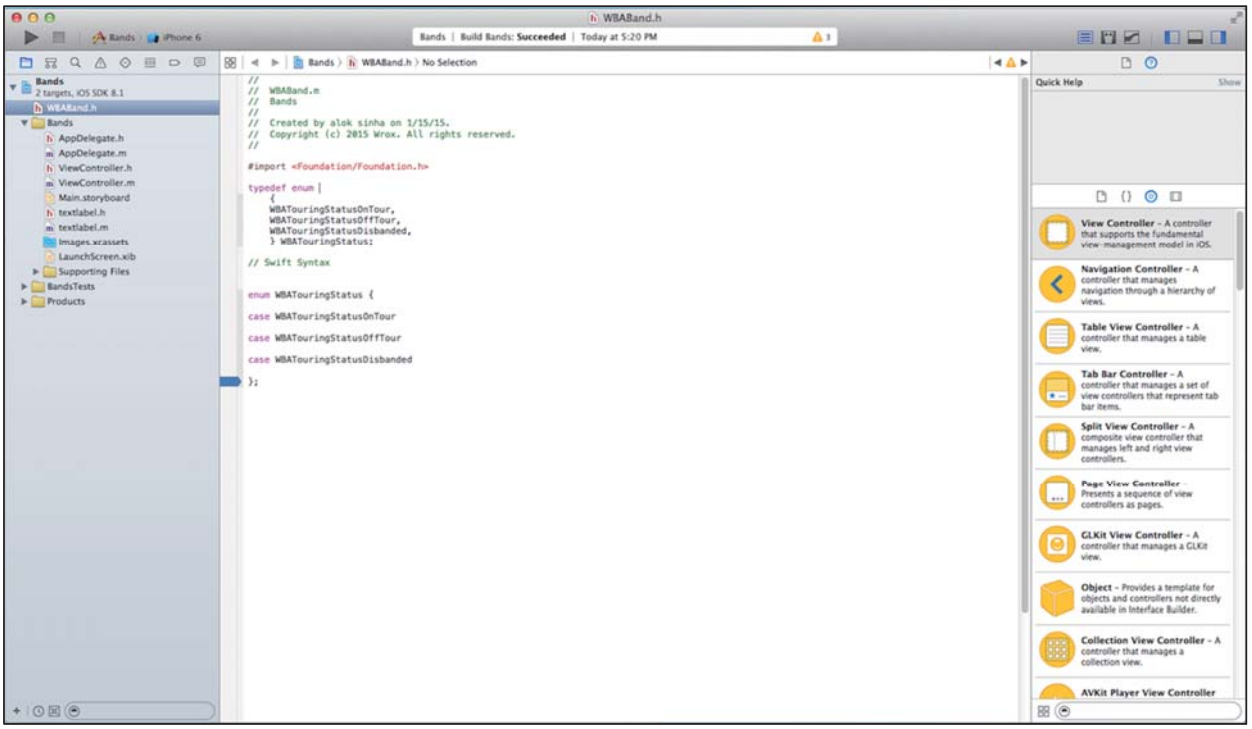

Figure 6: Write Code after the Import Section

3. Save the file and compile the application by selecting **Project** ➪ **Build** from the menu to ensure that there are no errors.

Unlike Objective-C, enumeration members in Swift are not assigned a default integer value when they are created.

Adding Properties to the Bands Model Object

After declaring the `WBATouringStatus` enumeration, you will have all the types you need to add to the properties of the `WBABand` class to the code. For the properties to be accessible in the code, you will add them as properties to the `WBABand.h` class.

Steps for Adding Properties to a Class

1. Select the `WBABand.h` option in the Project Navigator.
2. Add the following code to the interface:

```
@property (nonatomic,strong) NSString *name;
@property (nonatomic, strong) NSString *notes;
@property (nonatomic, assign) int rating;
@property (nonatomic, assign) WBATouringStatus touringStatus;
@property (nonatomic, assign) BOOL haveSeenLive;
```

In Swift, name and notes will be declared with the `var` keyword as they are mutable. You will write the code as follows:

```
var name: String? = nil
var notes: String? = nil
var rating: int
```

3. Save the file and compile the application by selecting **Project** ➪ **Build** from the menu to ensure that there are no errors.

Properties enable you to add member variables to Objective-C classes and also in Swift. The code you add creates all the member variables for the `WBABand` class. By declaring the enumeration types, you can declare a property using that type in the class interface.

ADDITIONAL KNOWHOW

The typical naming convention for enumerations is to start with the same prefix abbreviation you are using for class names followed by the name of the enumeration type with the differentiating value at the end. This helps the readability of your code both for yourself and for any other developer who may work on the application. By default, the elements are assigned their integer values based on their placement in the list.

BUILDING AN INTERACTIVE USER INTERFACE

The properties of `UIView` are set by using the Attributes inspector in Xcode. This is known as setting properties at design time. All user interface objects can be created and set this way, but they cannot be changed in code.

Creating the user interface (UI) of an app is itself a great deal. However, despite attractive and effective features, if a user does not find the UI of an app effective as per his/her expectation, he/she will be reluctant to use the app and might even uninstall it. In other words, the UI of an app is itself a great acceptance to the user and your app will get scores and rewards accordingly. A great UI will help the users to understand the app easily. For example, successful ecommerce apps such as **Amazon** and **Flipkart** have a UI that makes it easier for the customer to buy anything.

With apps, any business organization can stay in touch with its customers by sending messages and notifications for special events, promotions, and upcoming sales offers. The trick is to make these apps more user-friendly, relevant, and realistic.

Learning About `IBOutlet`

`IBOutlet` stands for Interface Builder Outlet. Xcode uses an `IBOutlet` instance variable to connect objects in a code to objects in the user interface. With the MVC design pattern, the `UIViewController` controls the `UIView`. The **Single View Application**

template you used to create the **Bands** project includes the `ViewController`, which is set as the `UIViewController` for the `UIView` in the Storyboard.

Steps for Connecting an `IBOutlet`

1. In Xcode, drag a `UILabel` onto the `UIView`. Use the Interface Builder guidelines to align the `UILabel` at the top and center of the `UIView`. Then, set its text to **Bands**, as shown in **Figure 7**.

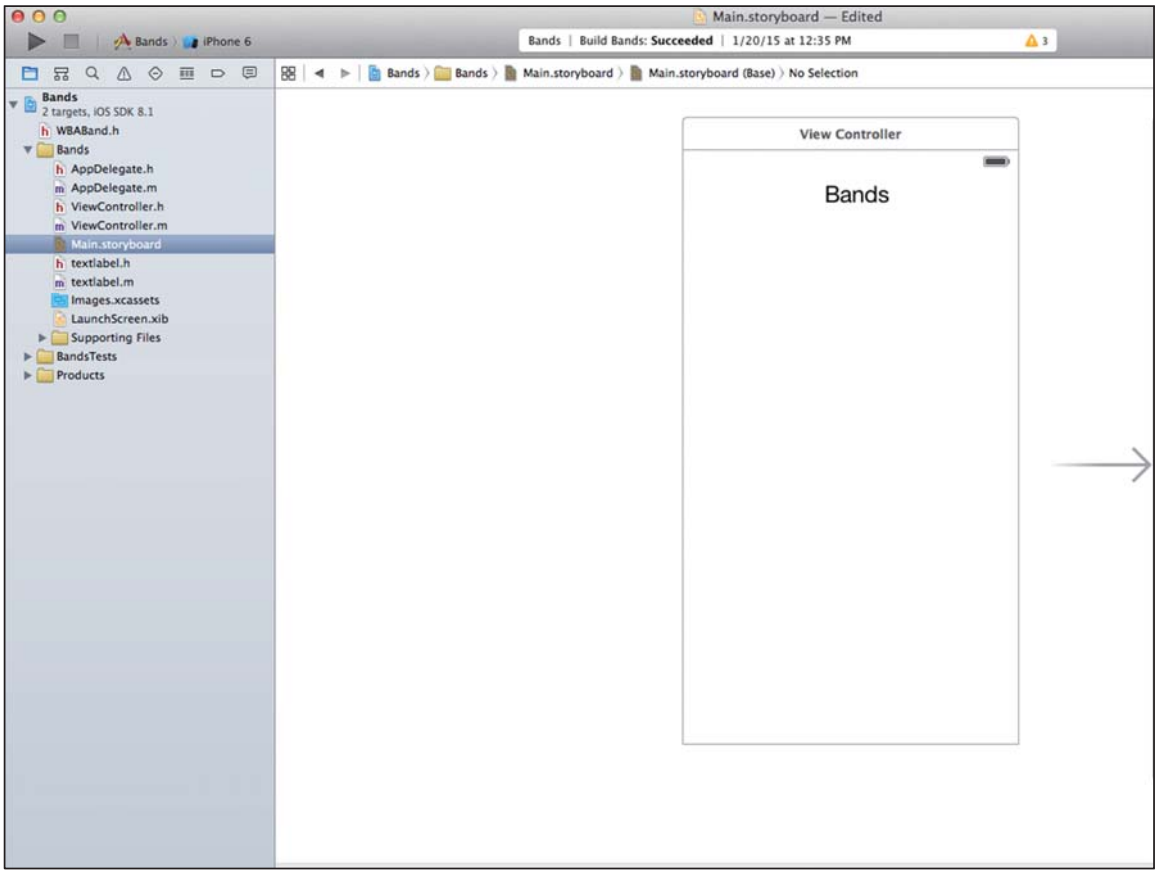

Figure 7: Show Bands in `UIView`

2. Select the `ViewController.h` file from the Project Navigator and add the following code to the interface section:

```
@interface ViewController : UIViewController
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@end
```

In Swift, add the code as follows:

```
class ViewController: UIViewController {
@IBOutlet weak var titleLabel: UILabel!
}
```

3. Return to the `Main.storyboard`, and select the View Controller from the Storyboard hierarchy on the left of the Editor.
4. Control-drag the line that is shown to the `UILabel` until both the View Controller and the `UILabel` are highlighted and connected.

5. The View Controller is connected to `UIButton` name as **Bands** (see **Figure 8**).

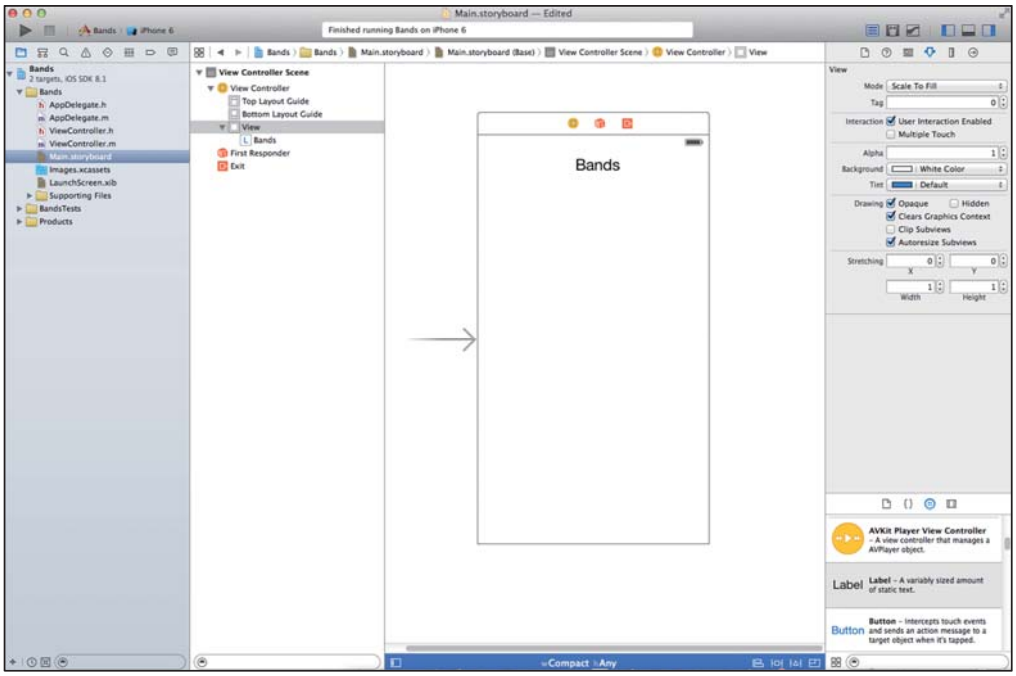

Figure 8: Connect View Controller to the `UIButton` Name as Bands

6. Release the mouse button and then select `titleLabel` from the Outlets dialog box, as shown in **Figure 9**.

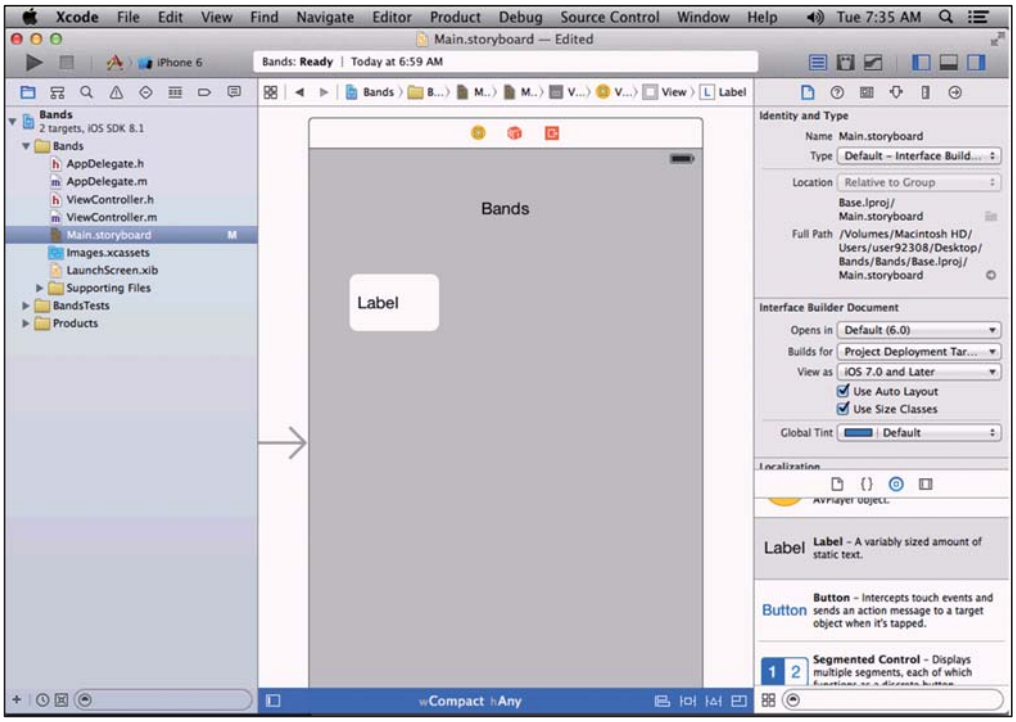

Figure 9: Select `TitleLabel`

7. Select the `ViewController.m` file from the Project Navigator.
8. Add the following code to the `viewDidLoad` method:

```
- (void)viewDidLoad
{
[super viewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
NSLog(@"titleLabel.text = %@", self.titleLabel.text);
}
```

In Swift, add the following code to the `viewDidLoad` method:

```
override func viewDidLoad()
{
                super.viewDidLoad()
 // Do any additional setup after loading the view, typically from a nib.
NSLog(@"titleLabel.text = %@", self.titleLabel.text);
}
```

9. Run the application in the iOS simulator. This will display `titleLabel.text = Bands` in the console, as shown in **Figure 10**.

Figure 10: iOS Simulator Showing Text UI as Bands

ADDITIONAL KNOWHOW

When referring to user interface (UI) objects, this session will use the names you see in Xcode where appropriate, but otherwise you will refer to them by their `UIKit` names. For instance, when you add a new UI object from the Objects library or interact with the Storyboard hierarchy, the UI object is labeled by common names, such as Label or Text field in Xcode. In those situations, the session will refer to them as Label or Text field. In most other situations, they will be referred to by their `UIKit` names.

`IBOutlet` properties are always created as weak properties. This is because the Storyboard is the owner of the object. The code only needs a weak reference to the object.

Using `UITextField` and `UITextFieldDelegate`

`UILabel` is one of the most basic `UIKit` objects. However, the **Bands** app needs to enable users to type in text of their own for a band name. For a single line of text, you use the `UITextField` control. Keeping with the MVC design pattern, the `UITextField` will ask its controller how it should act. It does this using the `UITextFieldDelegate` protocol. In the **Bands** app, the controller for the `UITextField` is the `ViewController` class, so it needs to implement the `UITextFieldDelegate` protocol.

Steps for Adding a `UITextField`

1. In the `Main.storyboard` file, add a new `UILabel` to the `UIView` and use the Interface Builder guidelines to align it to the left side of the `UIView`. Then set its text to **Name:**, as shown in **Figure 11**.
2. Find and drag a new Text field from the Objects library to the `UIView`, and align it under the **Name:** `UILabel`. Stretch it to the left and right guidelines of the `UIView`, as shown in **Figure 11**.

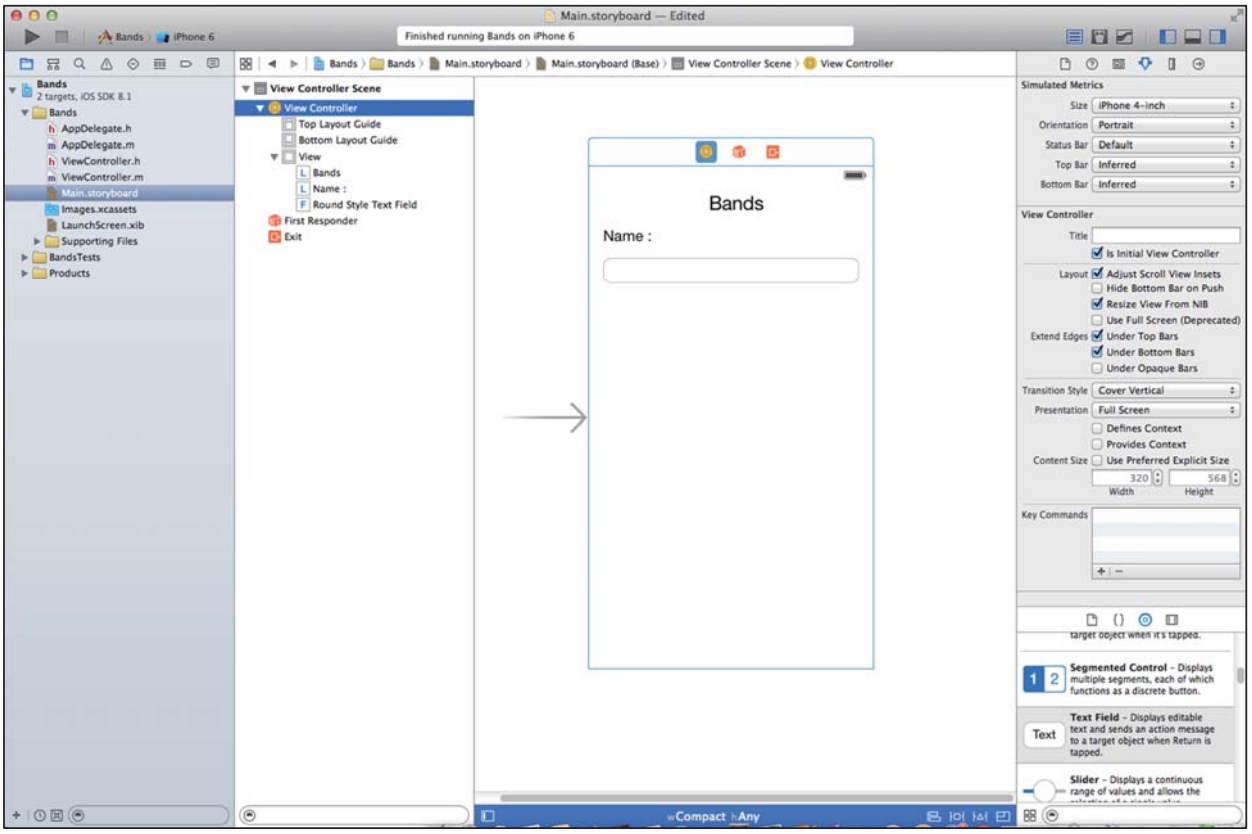

Figure 11: Drag a Text Field to the `UIView` under the Name: `UILabel`

3. Select `ViewController.h` from the Project Navigator and add the following code to the interface:

```
#import "WBABand.h"
@interface ViewController : UIViewController <UITextFieldDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@end
```

In Swift, you can use `ViewController` and other classes as follows:

```
class ViewController: UIViewController{
var bandObject:WBABand!
IBOutlet weak var titleLabel: UILabel!
IBOutlet weak var nameTextField: UITextField!
}
```

4. Return to the `Main.storyboard` file and select View Controller from the Storyboard hierarchy on the left of the Editor.
5. Connect the `nameTextField` to the `UITextField` using the same steps as indicated in the previous section, *Steps for Connecting an `IBOutlet`*.
6. Select the `UITextField` in the `UIView`. Control-drag the line back to the View Controller in the Storyboard hierarchy and select the delegate in the dialog box that opens, as shown in **Figure 12** and **Figure 13**.

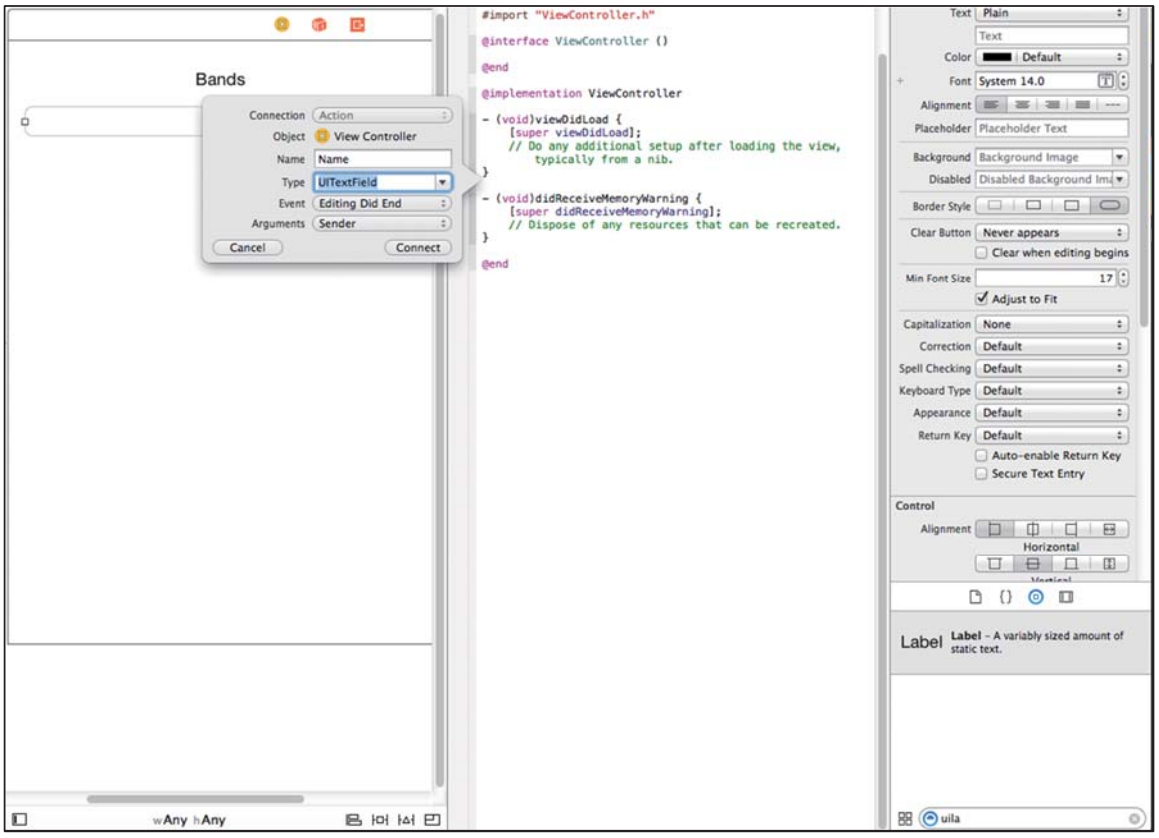

Figure 12: Select the `UITextField` in the `UIView`

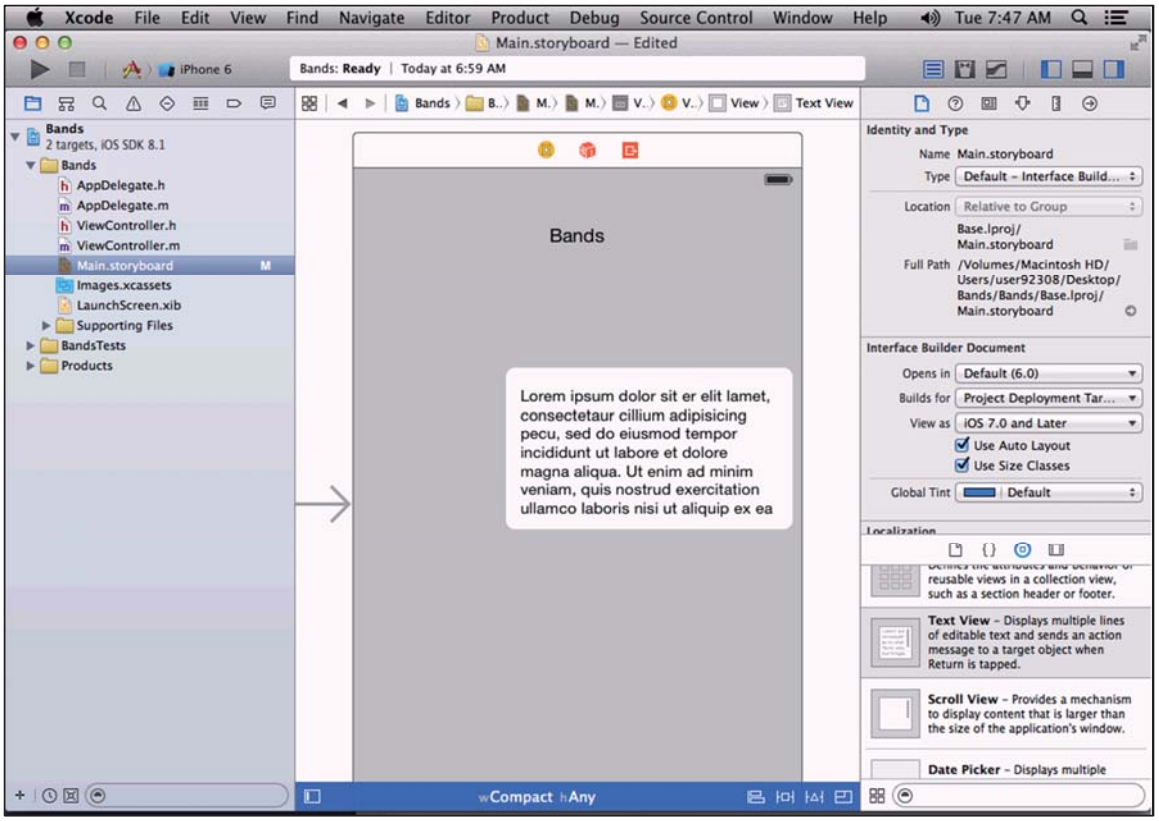

Figure 13: `UITextField` in the `UIView`

7. Select the `ViewController.m` file from the Project Navigator and add the following code to the `viewDidLoad` method:

```
- (void)viewDidLoad
{
[super viewDidLoad]; // Do any additional setup after loading the view, typically from a nib.
NSLog(@"titleLabel.text = %@", self.titleLabel.text);
}
self.bandObject = [[BandObject alloc] init];
```

8. Add the following code to the implementation of `UITextField`:

```
- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField
{
return YES;
}
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
self.bandObject.name = self.nameTextField.text;
[self.nameTextField resignFirstResponder];
return YES;
}
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField
```

```
{
self.bandObject.name = self.nameTextField.text;
[self saveBandObject];
[self.nameTextField resignFirstResponder];
return YES;
}
```

9. Run the application on the iPhone 4-inch Simulator and select the `UITextField`. The software keyboard appears, as shown in **Figure 14**.

Figure 14: Run App in iPhone 4-Inch Simulator

10. Enter **My Band** in the `UITextField` and tap the **return** key on the software keyboard. This will enter the **My Band** text into the `UITextField` and hide the software keyboard.

If the keyboard does not disappear after you have pressed the **return** key, check the following points:

- Whether you have implemented the `textFieldShouldReturn:` method
- Whether the `textFieldShouldReturn:` method resigns the `nameTextField` as the first responder
- Whether the delegate of the `nameTextField` has been connected to the `ViewController` class

If you miss either of these actions, it will cause the keyboard to remain on the screen.

Using `UITextView` and `UITextViewDelegate`

For the **Bands** app, each band has notes associated with it where the user can type. These notes can be multiple lines. For text that needs multiple lines, you use a `UITextView` object. A `UITextView` object is similar in implementation to a `UITextField` object. It asks its controller how it should act through the `UITextViewDelegate` protocol. Just like the `UITextField`, the `ViewController` class will be a delegate for the `UITextView` object.

Steps for Adding a `UITextView`

1. In the `Main.storyboard` file, add a new `UILabel` to the `UIView` using the Interface Builder guidelines to align it to the left side of the `UIView`. Then, set its text to **Notes**, as shown in **Figure 15**.

2. Find and drag a new Text field from the Objects library onto the `UIView`. Align it under the **Notes** `UILabel` and stretch it to the left and right guidelines of the `UIView`. Then, set its height to 90 pixels, as shown in **Figure 15**.

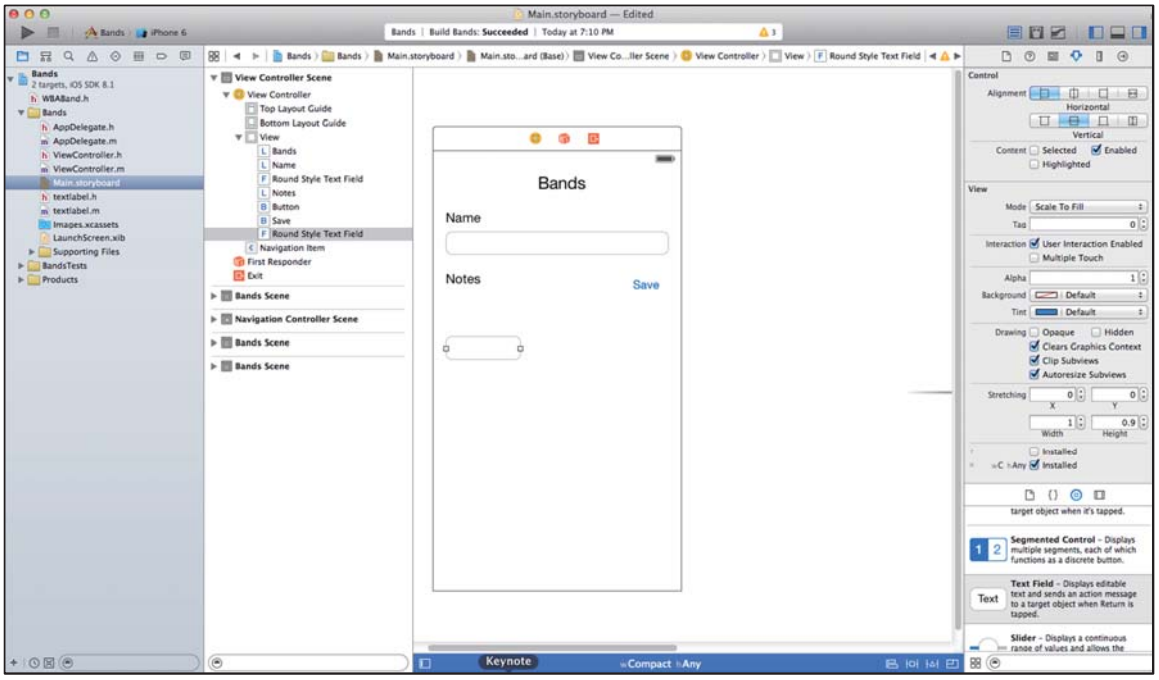

Figure 15: Drag the Text Field under Notes

3. In the Attributes inspector, change the background color of the `UITextView` to light gray, as shown in **Figure 16**.

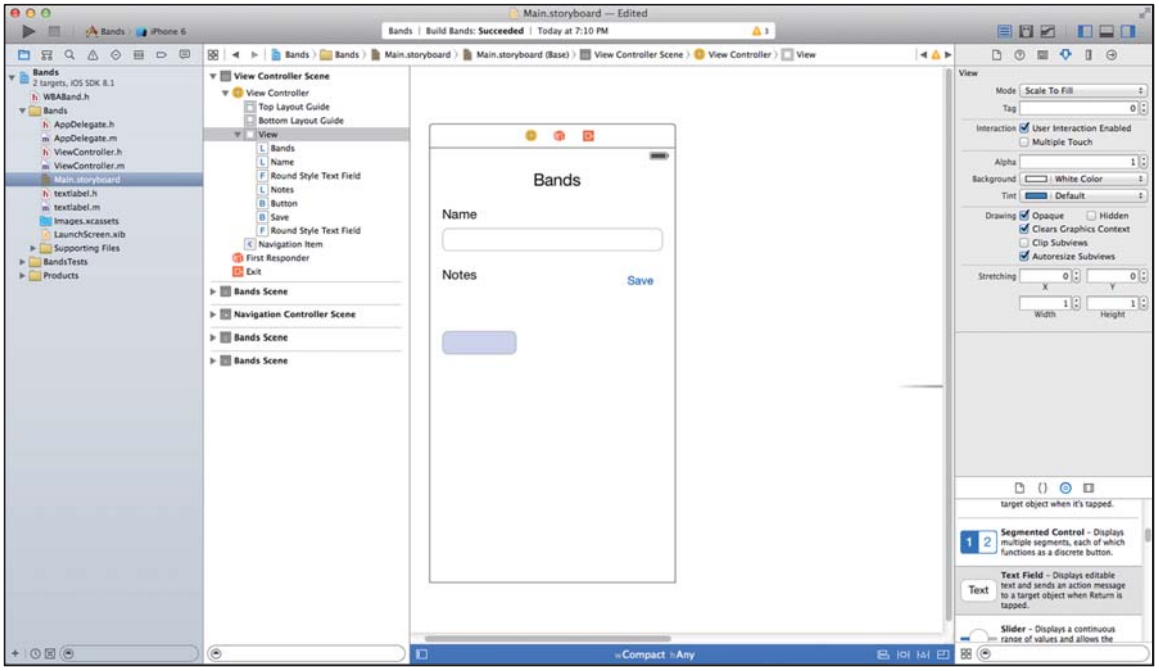

Figure 16: Change the `TextView` Background

4. Select `ViewController.h` from the Project Navigator and add the following code to the interface:

```
@interface ViewController : UIViewController <UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@end
```

5. Return to the `Main.storyboard` and select the View Controller from the Storyboard hierarchy on the left side of the Editor.
6. Connect the `notesTextView` to the `UITextView` by following the same steps as in the previous section.
7. Connect the delegate of the `notesTextView` to the `ViewController` using the same steps as in the previous section.
8. Select the `ViewController.m` file from the Project Navigator and add the following code to the implementation of `UITextView`:

```
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
{
return YES;
}
- (BOOL)textViewShouldEndEditing:(UITextView *)textView
{
self.bandObject.notes = self.notesTextView.text;
[self.notesTextView resignFirstResponder];
return YES;
}
```

9. Run the application in the iPhone 4-inch simulator. When you tap the `notesTextView`, the keyboard appears and enables you to enter text, as shown in **Figure 17**.

Figure 17: Run App in iOS Simulator

10. Tap the **return** button to add a line break to the text.

Using UIButton and IBAction

The simplest way to tell the UITextView that you are done entering text is to add a UIButton, which when touched resigns the UITextView as the first responder. A UIButton, however, does not have a corresponding UIButtonDelegate. Instead, you have to connect the touch event to a method in your code. To make that method visible in Interface Builder, you use the IBAction macro, which is short for Interface Builder Action.

Steps for Adding a UIButton

1. Select the Main.storyboard file from the Project Navigator to open Interface Builder.
2. Find and drag a new Button from the Objects library and align it on the right side of the UIView in line with the **Notes** UILabel, as shown in **Figure 18**.
3. In the Attributes inspector, change the text of the UIButton to **Save** (see **Figure 18**) and deselect the **Enabled** check box.

Figure 18: Change UIButton Name to Save

4. Select ViewController.h from the Project Navigator and add the following code to the interface:

```
@interface ViewController : UIViewController <UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
- (IBAction)saveNotesButtonTouched:(id)sender;
@end
```

In Swift, this format is as follows:

```
//--Format in Swift---
class ViewController: UIViewController {
var delegate : AppDelegate
}

//---Using this Format swift code will be ---
class ViewController: UIViewController{
   var bandObject:WBABand!
   IBOutlet weak var titleLabel: UILabel!
IBOutlet weak var nameTextField: UITextField!
IBOutlet weak var nameTextView: UITexView!
IBOutlet weak var saveNotesButton: UIButton!
}
@IBAction func saveNotesButton(sender: UIButton) {
    }
```

5. Select `ViewController.m` and add the following code to the implementation of `UITextView`:

```
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
{
self.saveNotesButton.enabled = YES;
return YES;
}
- (BOOL)textViewShouldEndEditing:(UITextView *)textView
{
self.bandObject.notes = self.notesTextView.text;
[self.notesTextView resignFirstResponder];
self.saveNotesButton.enabled = NO;
return YES;
}
- (IBAction)saveNotesButtonTouched:(id)sender
{
[self textViewShouldEndEditing:self.notesTextView];
}
```

6. Return to the `Main.storyboard` and connect the `saveNotesButton` to the `UIButton`.
7. Control-drag back to the View Controller in the Storyboard hierarchy and select `saveNotesButtonTouched:` from the dialog box, as shown in **Figure 19**.

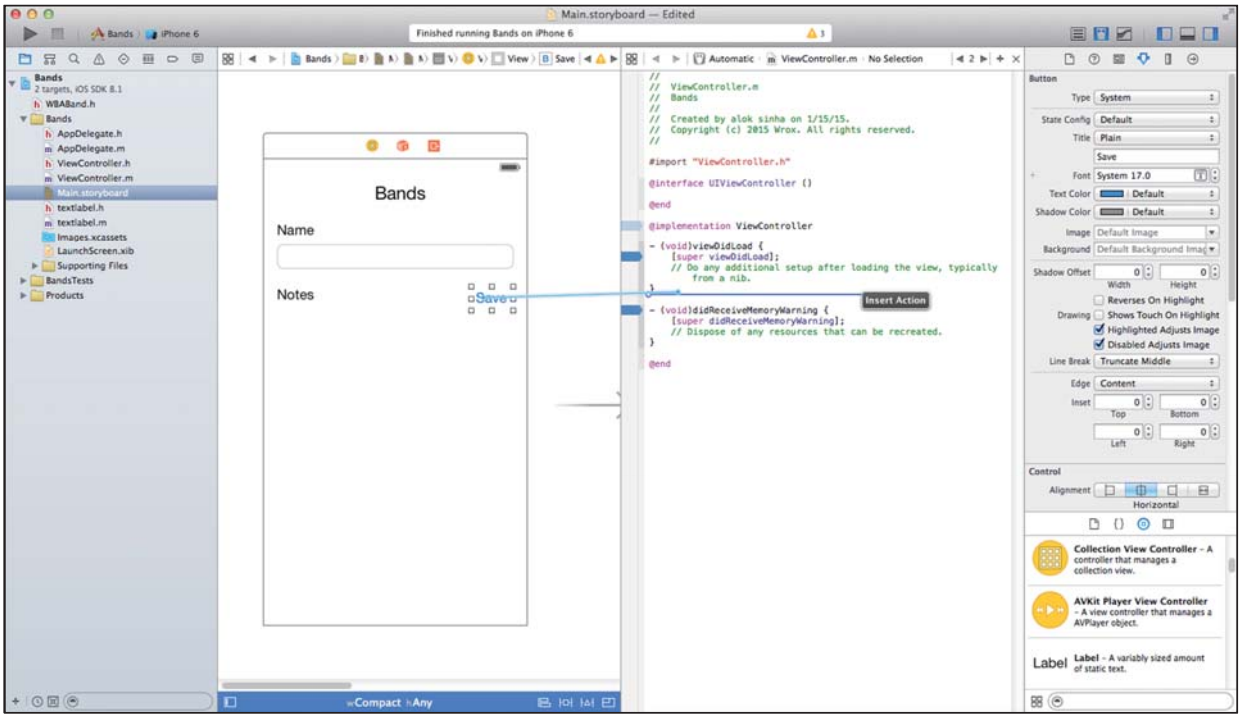

Figure 19: Control-Drag back to the View Controller

8. Run the application in the iPhone 4-inch simulator (see **Figure 20**). When you select the `notesTextView`, the `saveNotesButton` is enabled. When you tap the `saveNotesButton`, it is disabled and the keyboard is hidden.

Figure 20: Run the App in Simulator, which Shows the Bands `UIView`

Using UIStepper

While building a user interface, you might need a user interface object that increases or decreases an integer. You can use the UIStepper element for this. UIStepper is a simple control with a minus button on the left and a plus button on the right. Tapping either button adjusts its value up or down. You can use UIStepper to represent the rating property of the bandObject.

Steps for Adding a UIStepper

1. In the Main.storyboard, add a new UILabel and use the Interface Builder guidelines to align it on the left side of the UIView. Then, set its text to **Rating**, as shown in **Figure 21**.

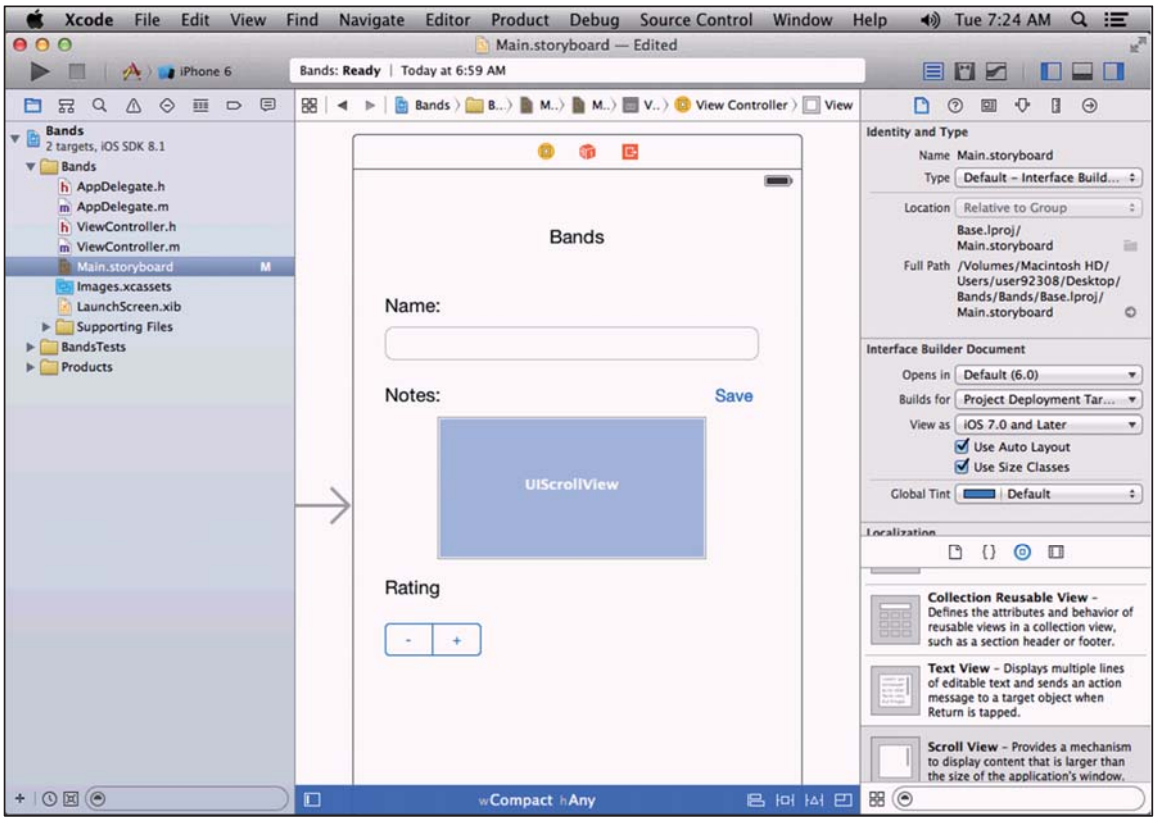

Figure 21: Add UILabel to UIView and Set Text to Rating

2. Find and drag a new Stepper from the Objects library, and use the Interface Builder guidelines to align it on the left side of the UIView underneath the **Rating** UILabel.
3. In the Attributes inspector, set the minimum value to 0, the maximum value to 10, the current value to 0, and the step value to 1.
4. Add another UILabel to the UIView, and align it on the Interface Builder guidelines to the right side of the UIView and vertically centered with the UIStepper. Then, set its text to 0, as shown in **Figure 22**.

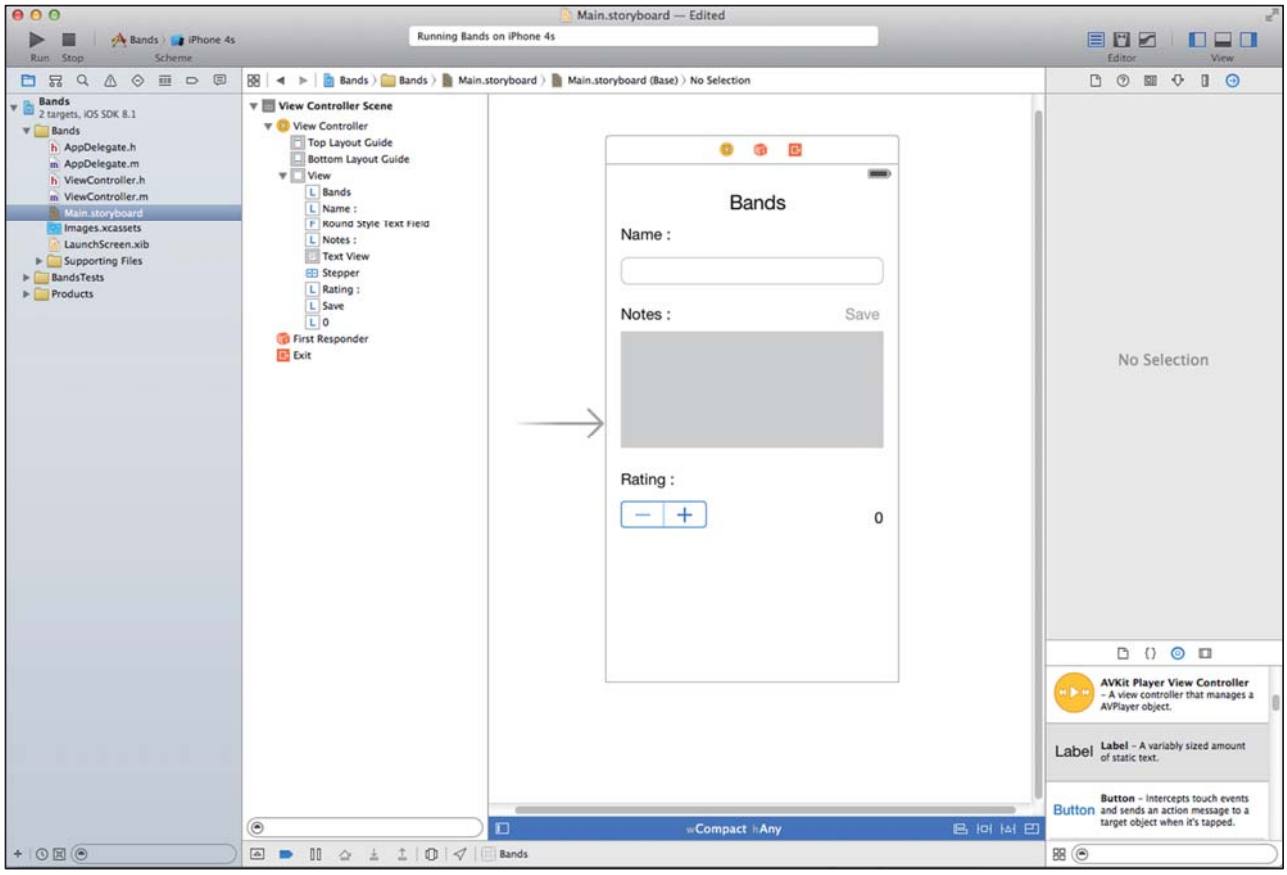

Figure 22: Add `UILabel` to `UIView` and Align it on Interface Builder

5. Select `ViewController.h` from the Project Navigator and add the following code to the interface:

```
@interface ViewController : UIViewController <UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
@property (nonatomic, weak) IBOutlet UIStepper *ratingStepper;
@property (nonatomic, weak) IBOutlet UILabel *ratingValueLabel;

-(IBAction)saveNotesButtonTouched:(id)sender;
-(IBAction)ratingStepperValueChanged:(id)sender;
@end
```

6. Select `ViewController.m` and add the following code to the implementation:

```
- (IBAction)ratingStepperValueChanged:(id)sender

{
self.ratingValueLabel.text =
```

```
    [NSString stringWithFormat:@"%g", self.ratingStepper.value];
    self.bandObject.rating = (int)self.ratingStepper.value;
    }
```

7. Return to `Main.storyboard`, and connect the `ratingStepper` to the `UIStepper` and the `ratingValueLabel` to `UILabel` on the right side of the `UIView`.
8. Connect the `ratingStepperValueChanged:` method to the `ratingStepper`.
9. Run the application in the iPhone 4-inch simulator. When you tap the plus and minus buttons of the `ratingStepper`, its value displays in the `ratingValueLabel`, as shown in **Figure 23**.

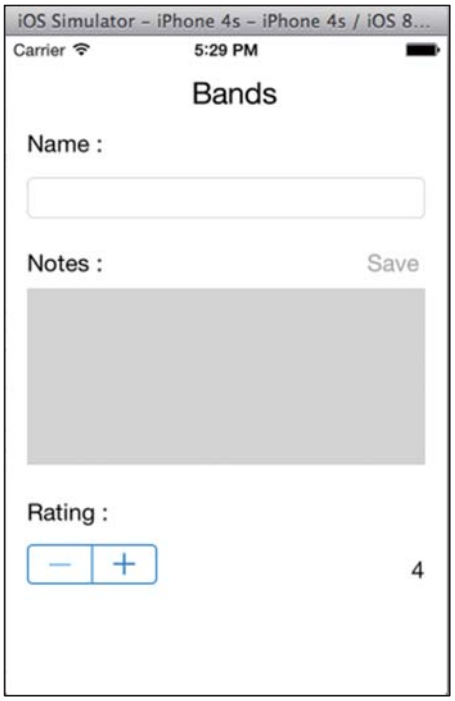

Figure 23: Run Application in iPhone 4-Inch Simulator

Using `UISegmentedControl`

The next property of the `WBABand` class that you add to the interface is the **Touring Status**. For this, you use a `UISegmentedControl` object. The `UISegmentedControl` object has the same look as the `UIStepper` object you added in the last section, except that you can control the number of its segments and the text or image of those segments. Each segment acts as its own button and can either stay selected when tapped or it can be "momentary" like the buttons in the `UIStepper` control. For the **Touring Status** option, use a `UISegmentedControl` object that stays selected, as shown in **Figure 24**.

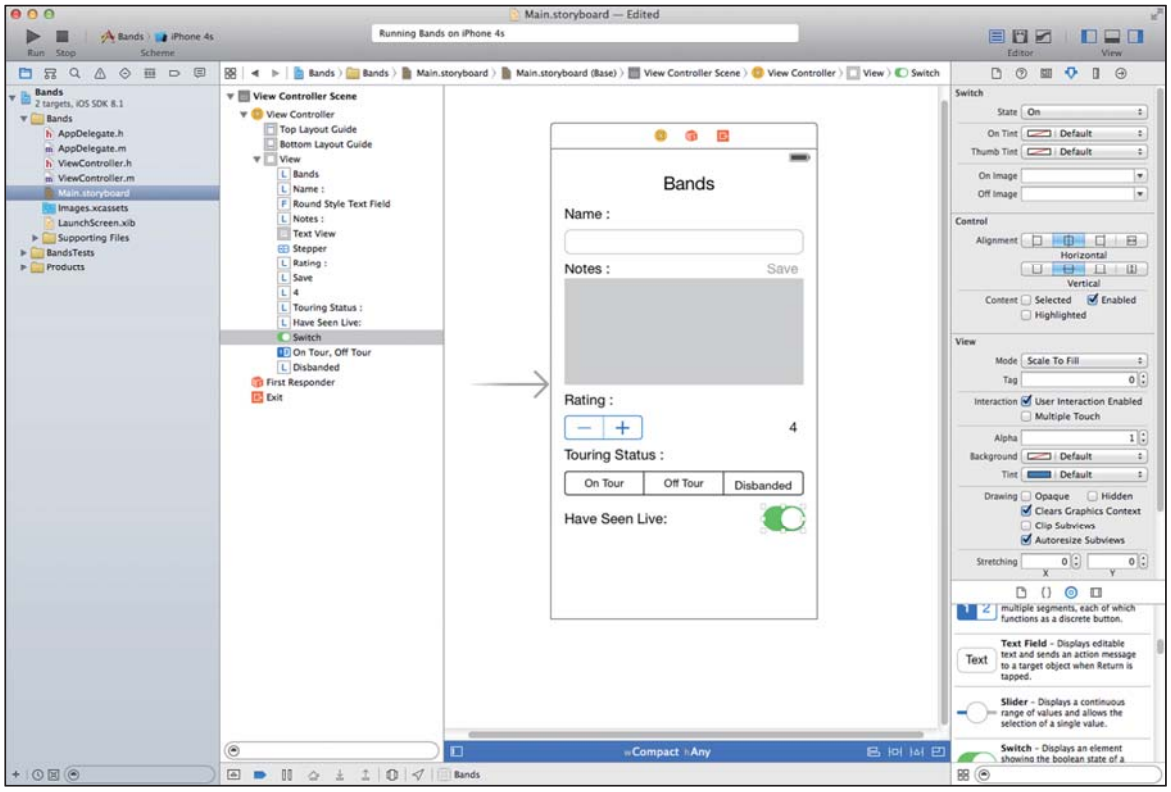

Figure 24: Touring Status of the `UISegmentedControl` Object

Using `UISwitch`

The last property of the `WBABand` class you need to add to the user interface is the `haveSeen` property using a `UISwitch` object, as shown in **Figure 24**. The `UISwitch` object is what you would expect it to be—a user interface object that is either on or off. It also does not have a corresponding delegate, so you need one more `IBAction` method to know when the user interacts with it.

Steps for Adding a `UISwitch`

1. In `Main.storyboard`, add a new `UILabel` using the Interface Builder guidelines to align it on the left side of the `UIView`. Then, set its text to **Have Seen Live**, as shown in **Figure 24.**
2. Find and drag a new Switch from the Objects library and add it to the `UIView`, aligning it with the vertical center of the **Have Seen Live** `UILabel` and the right side of the `UIView`.
3. Select `ViewController.h` from the Project Navigator and add the following code to the interface:

```
@interface ViewController: UIViewController<UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
@property (nonatomic, weak) IBOutlet UIStepper *ratingStepper;
@property (nonatomic, weak) IBOutlet UILabel *ratingValueLabel;
@property (nonatomic, weak) IBOutlet UISegmentedControl
*touringStatusSegmentedControl;
```

```
@property (nonatomic, weak) IBOutlet UISwitch *haveSeenLiveSwitch;
- (IBAction)saveNotesButtonTouched:(id)sender;
- (IBAction)ratingStepperValueChanged:(id)sender;
- (IBAction)tourStatusSegmentedControlValueChanged:(id)sender;
- (IBAction)haveSeenLiveSwitchValueChanged:(id)sender;
@end
```

In Swift, each `IBAction` button in `ViewController.swift` is made as a function for the task. To do so, you can use the following code:

```
@IBAction func saveNotesButtonTouched(sender: UIButton){
}
@IBAction func ratingStepperValueChanged(sender: UIButton){
}
@IBAction func tourStatusSegmentedControlValueChanged(sender: UIButton){
}
@IBAction func haveSeenLiveSwitchValueChanged(sender: UIButton){
}
```

4. Select `ViewController.m` from the Project Navigator and add the following code to the implementation:

```
- (IBAction)haveSeenLiveSwitchValueChanged:(id)sender
{
self.bandObject.haveSeenLive = self.haveSeenLiveSwitch.on;
}
```

5. Return to `Main.Storyboard`, and connect the `haveSeenLiveSwitch` property and the `haveSeenLiveSwitchValueChanged:` method to the `UISwitch`.
6. Run the application in the iPhone 4-inch simulator. You can toggle the `haveSeenLiveSwitch` off and on to change the value of the `haveSeenLive` property of the `bandObject`, as shown in **Figure 25**.

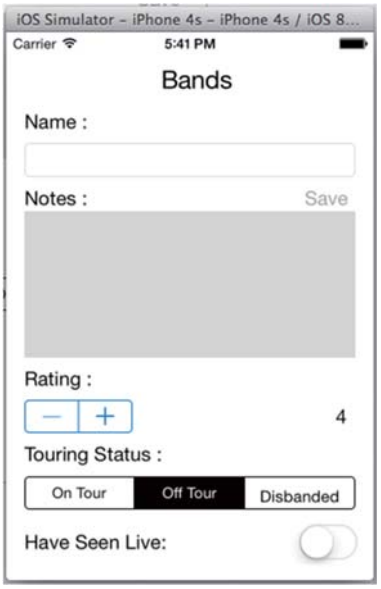

Figure 25: Run the Application in iPhone 4-Inch Simulator

ADDITIONAL KNOWHOW

`UIButton` is a `UISegmentedControl` that does not have a corresponding delegate protocol. Therefore, to know when a user interacts with it, you add an `IBAction` method named `tourStatusSegmentedControlValueChanged:`. It gets triggered when the selected segment changes. In its implementation, you can use the `selectedSegmentIndex` property to set the `touringStatus` property of the `bandObject` because both the segments and the `WBATouringStatus` enumeration are 0-based.

SAVING AND RETRIEVING DATA

Giving the users the ability to enter data into your app is not enough. This ability is only useful if users are able to save and retrieve data. There are many ways to do this in an iOS application, but the simplest is to use `NSUserDefaults`. The documented use for `NSUserDefaults` is to save preferences for an application. However, because of its simplicity, `NSUserDefaults` is often used to save small amounts of data as well. You can use it to save and retrieve an instance of the `WBABand` class.

Implementing the `NSCoding` Protocol

Before saving an instance of the `WBABand` class, you need to declare that this class will implement the `NSCoding` protocol. You then need to add the protocol's two methods, `initWithCoder:` and `encodeWithCoder:`, to the `WBABand` class implementation. These methods give the class a way to encode and decode itself so that it can be archived and saved to persistent storage.

1. Select the `WBABand.h` file from the Project Navigator and add the following code to the interface:

```
@interface WBABand : NSObject <NSCoding>
@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *notes;
@property (nonatomic, weak) int rating;
@property (nonatomic, weak) WBATouringStatus touringStatus;
@property (nonatomic, weak) BOOL haveSeenLive;
@end
```

2. Select the `WBABand.m` file from the Project Navigator and add the following code to the implementation:

```
static NSString *nameKey = @"BANameKey";
static NSString *notesKey = @"BANotesKey";
static NSString *ratingKey = @"BARatingKey";
static NSString *tourStatusKey = @"BATourStatusKey";
static NSString *haveSeenLiveKey = @"BAHaveSeenLiveKey";
@implementation WBABand
-(id) initWithCoder:(NSCoder*)coder
{
self = [super init];
self.name = [coder decodeObjectForKey:nameKey];
self.notes = [coder decodeObjectForKey:notesKey];
self.rating = [coder decodeIntegerForKey:ratingKey];
self.touringStatus = [coder decodeIntegerForKey:tourStatusKey];
self.haveSeenLive = [coder decodeBoolForKey:haveSeenLiveKey];
}
```

```
return self;
- (void)encodeWithCoder:(NSCoder *)coder
{
[coder encodeObject:self.name forKey:nameKey];
[coder encodeObject:self.notes forKey:notesKey];
[coder encodeInteger:self.rating forKey:ratingKey];
[coder encodeInteger:self.touringStatus forKey:tourStatusKey];
[coder encodeBool:self.haveSeenLive forKey:haveSeenLiveKey];
}
@end
```

Saving Data

To save an instance of the `WBABand` class to persistent storage, you will use the `standardUserDefaults`, which is a global instance of the `NSUserDefaults` class. It works a bit like `NSCoder` in that it uses a key-value paring to save both primitive types and `NSObject` instances to disk.

To save an instance of the `WBABand` class, it first needs to be archived into an `NSData` object, which is an object-oriented wrapper around a byte buffer. To do this, you use the `NSKeyedArchiver` class, which is a subclass of `NSCoder`. When you call the `archiveDataWithRootObject:` method, it will call the `encodeWithCoder:` method you implemented in the `WBABand` class to create an `NSData` archive. The archive is then stored in the `standardUserDefaults` using the `setObject:forKey:` method.

Steps for Saving Data Using `NSUSerDefaults`

1. Select the `ViewController.h` file from the Project Navigator and add the following code to the interface:

```
@interface ViewController : UIViewController <UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
@property (nonatomic, weak) IBOutlet UIStepper *ratingStepper;
@property (nonatomic, weak) IBOutlet UILabel *ratingValueLabel;
@property (nonatomic, weak) IBOutlet UISegmentedControl
*touringStatusSegmentedControl;
@property (nonatomic, weak) IBOutlet UISwitch *haveSeenLiveSwitch;
- (IBAction)saveNotesButtonTouched:(id)sender;
- (IBAction)ratingStepperValueChanged:(id)sender;
- (IBAction)tourStatusSegmentedControlValueChanged:(id)sender;
- (IBAction)haveSeenLiveSwitchValueChanged:(id)sender;
- (void)saveBandObject;
@end
```

2. Select the `ViewController.m` file from the Project Navigator and add the following code before the implementation:

```
#import "ViewController.h"
static NSString *bandObjectKey = @"BABandObjectKey";
@implementation ViewController
```

3. Add the following code to the `ViewController` implementation:

```
- (void)saveBandObject
{
NSData *bandObjectData =
[NSKeyedArchiver archivedDataWithRootObject:self.bandObject];
[[NSUserDefaults standardUserDefaults]
setObject:bandObjectData forKey:bandObjectKey];
}
```

4. Add a call to the `saveBandObject` method after setting any of the properties in the previous methods:

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
self.bandObject.name = self.nameTextField.text;
[self saveBandObject];
[self.nameTextField resignFirstResponder];
return YES;
}

- (BOOL)textViewShouldEndEditing:(UITextView *)textView
{
self.bandObject.notes = self.notesTextView.text;
[self saveBandObject];
[self.notesTextView resignFirstResponder];
self.saveNotesButton.enabled = NO;
return YES;
}
- (IBAction)ratingStepperValueChanged:(id)sender
{
self.ratingValueLabel.text= [NSString stringWithFormat:@"%g",self.ratingStepper.value];
self.bandObject.rating = (int)self.ratingStepper.value;
[self saveBandObject];
}
- (IBAction)tourStatusSegmentedControlValueChanged:(id)sender
   {
self.bandObject.touringStatus =
self.touringStatusSegmentedControl.selectedSegmentIndex;
[self saveBandObject];
}
- (IBAction)haveSeenLiveSwitchValueChanged:(id)sender
{
self.bandObject.haveSeenLive = self.haveSeenLiveSwitch.on;
[self saveBandObject];
}
```

5. Build the project and make sure there are no errors.

Retrieving Saved Data

Retrieving stored data from `NSUserDefaults` is basically the reverse of saving. You retrieve an object from `standardUserDefaults` using the `objectForKey:` method and the same `bandObjectKey`. This returns the `NSData` archive. You then use the `unarchiveObjectWithData:` method of the `NSKeyedUnarchiver` class to unarchive the data and get back the `WBABand` instance.

Steps for Retrieving Data from `NSUserDefaults`

1. Select the `ViewController.h` file from the Project Navigator and add the following code to the interface:

```
@interface ViewController : UIViewController <UITextFieldDelegate,UITextViewDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
@property (nonatomic, weak) IBOutlet UIStepper *ratingStepper;
@property (nonatomic, weak) IBOutlet UILabel *ratingValueLabel;
@property (nonatomic, weak) IBOutlet UISegmentedControl
*touringStatusSegmentedControl;
@property (nonatomic, weak) IBOutlet UISwitch *haveSeenLiveSwitch;
- (IBAction)saveNotesButtonTouched:(id)sender;
- (IBAction)ratingStepperValueChanged:(id)sender;
- (IBAction)tourStatusSegmentedControlValueChanged:(id)sender;
-(IBAction)haveSeenLiveSwitchValueChanged:(id)sender;
-(void)saveBandObject;
-(void)loadBandObject;
-(void)setUserInterfaceValues;
@end
```

2. Select the `ViewController.m` file from the Project Navigator, and add the following code to the implementation:

```
- (void)loadBandObject
{
NSData *bandObjectData = [[NSUserDefaults standardUserDefaults]
objectForKey:bandObjectKey];
if(bandObjectData)
self.bandObject =
[NSKeyedUnarchiver unarchiveObjectWithData:bandObjectData];
}
- (void)setUserInterfaceValues
{
self.nameTextField.text = self.bandObject.name;
self.notesTextView.text = self.bandObject.notes;
self.ratingStepper.value = self.bandObject.rating;
self.ratingValueLabel.text = [NSString stringWithFormat:@"%g",
self.ratingStepper.value];
self.touringStatusSegmentedControl.selectedSegmentIndex =
self.bandObject.touringStatus;
```

```
self.haveSeenLiveSwitch.on = self.bandObject.haveSeenLive;
}
```

3. Modify the `viewDidLoad` method with the following code:

```
- (void)viewDidLoad
{
[super viewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
NSLog(@"titleLabel.text = %@", self.titleLabel.text);
[self loadBandObject];
if(!self.bandObject)
self.bandObject = [[BandObject alloc] init];
}
[self setUserInterfaceValues];
```

4. Run the application in the iPhone 4-inch simulator and enter some data.
5. Restart the application. The data you entered will be reloaded.

In the implementation of the `loadBandObject`, you first attempt to retrieve an `NSData` archive from the `standardUserDefaults` using the `bandObjectKey`. If there is no archive, this call will return `nil`. If there is an archive, the code will call the `unarchivedObjectWithData:` method of the `NSKeyedUnarchiver` class to unarchive the `WBABand` instance and set the `bandObject` property.

Deleting Saved Data

To delete data that you have saved to `standUserDefaults`, all you need to do is set the object for the key to `nil`. Adding **Delete** to the user interface, however, is a little more difficult. Before you delete any data, you need to verify with users that they actually want it deleted. The best way to do this in an iOS application is to use a `UIActionSheet`, which has a "destructive" button that will be red when presented to the user. This way the users know they are about to permanently delete the data.

A `UIActionSheet` also has its own delegate. When a user taps a button in a `UIActionSheet`, it tells its delegate about it using the `actionSheet:clickedButtonAtIndex:` method of the `UIActionSheetDelegate` protocol. For the **Bands** app, the `ViewController` class will act as the delegate so it needs to implement this protocol.

The `actionSheet:clickedButtonAtIndex:` method of the `UIActionSheetDelegate` protocol is used to know which button is clicked by the user.

Steps for Deleting Data from `NSUserDefaults`

1. In the `Main.storyboard` file, add a new `UIButton` to the bottom of the `UIView`, set its text to **Delete**, and add an auto layout constraint to anchor the button to the bottom of the view, as shown in **Figure 26**.

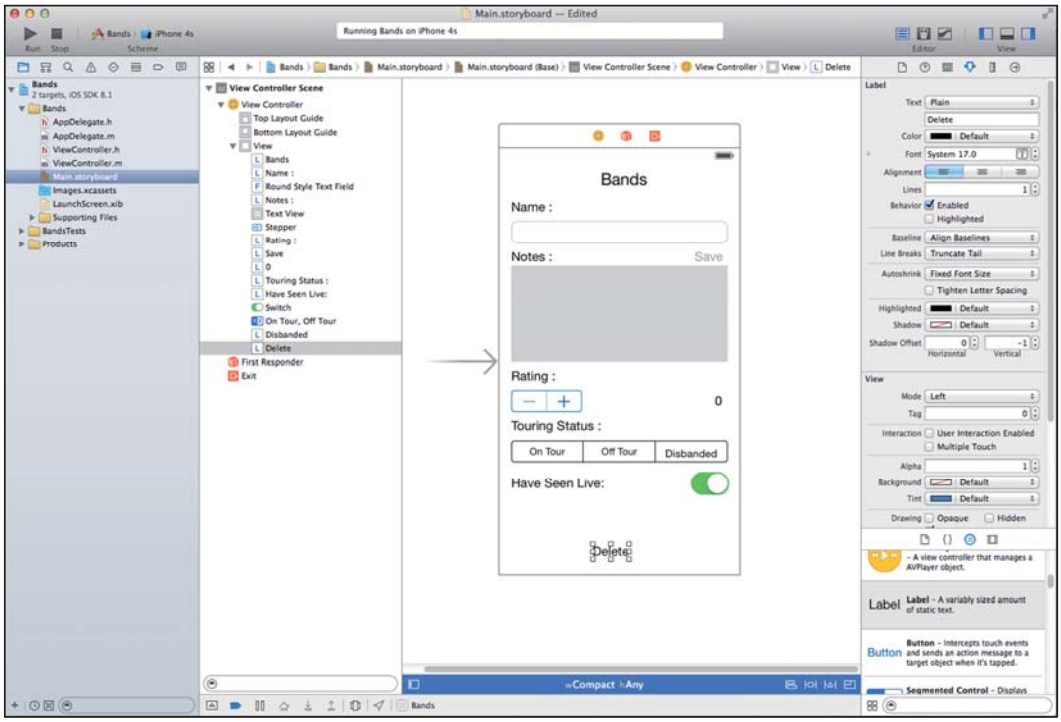

Figure 26: Add Delete to the `UIView`

2. Select `ViewController.h` from the Project Navigator and add the following code:

```
@interface ViewController : UIViewController <UITextFieldDelegate,
UITextViewDelegate, UIActionSheetDelegate>
@property (nonatomic, strong) WBABand *bandObject;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UITextField *nameTextField;
@property (nonatomic, weak) IBOutlet UITextView *notesTextView;
@property (nonatomic, weak) IBOutlet UIButton *saveNotesButton;
@property (nonatomic, weak) IBOutlet UIStepper *ratingStepper;
@property (nonatomic, weak) IBOutlet UILabel *ratingValueLabel;
@property (nonatomic, weak) IBOutlet UISegmentedControl
*touringStatusSegmentedControl;
@property (nonatomic, weak) IBOutlet UISwitch *haveSeenLiveSwitch;
-(IBAction)saveNotesButtonTouched:(id)sender;
- (IBAction)ratingStepperValueChanged:(id)sender;
- (IBAction)tourStatusSegmentedControlValueChanged:(id)sender;
- (IBAction)haveSeenLiveSwitchValueChanged:(id)sender;
- (IBAction)deleteButtonTouched:(id)sender;
- (IBAction)deleteButtonTouched:(id)sender;
- (void)saveBandObject;
- (void)loadBandObject;
- (void)setUserInterfaceValues;
@end
```

2. Select the `WBABand.m` file from the Project Navigator and add the following code to the implementation:

```
- (IBAction)deleteButtonTouched:(id)sender
{
UIActionSheet *promptDeleteDataActionSheet = [[UIActionSheet alloc]
initWithTitle:nil delegate:self cancelButtonTitle:@"Cancel"
destructiveButtonTitle:@"Delete Band" otherButtonTitles:nil];
[promptDeleteDataActionSheet showInView:self.view];
}
- (void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
if(actionSheet.destructiveButtonIndex == buttonIndex)
{
self.bandObject = nil;
[self setUserInterfaceValues];
}
}
[[NSUserDefaults standardUserDefaults] setObject:nil forKey:bandObjectKey];
```

3. Run the application in the iPhone 4-inch simulator. When you tap the **Delete** button, the `UIActionSheet` will appear asking if you want to delete the band, as shown in **Figure 27**. Tapping the **Delete Band** option will delete the data from `standardUserDefaults`.

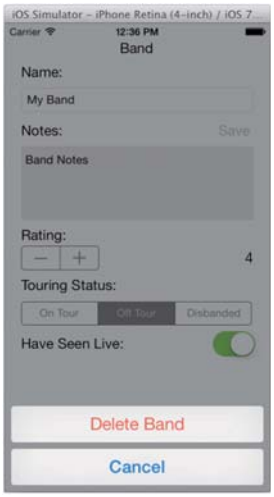

Figure 27: Run the Application in iPhone 4-Inch Simulator

ADDITIONAL KNOWHOW

The `standardUserDefaults` object saves to a disk at periodic intervals. Therefore, you do not need to do anything more to get the object written to a disk.

During the lab hour of this session, you will create a user input form by using a static table in the Storyboard.

Cheat Sheet

- The first step to build an iOS application is to create a class that will represent the model project of your app.
- The next step is to declare an enumeration to represent the types you need to add to the properties of your app.
- Then, you add properties to add member variables to your app class.
- To build an interactive user interface of your app, you create and set the following UI objects:
  - `IBOutlet` to connect objects in a code to objects in the user interface
  - `UITextField` to add a single line of text to the user interface
  - `UITextView` to add notes or multiple lines of text to the user interface
  - `UIButton` to tell the `UITextView` that you are done entering text
  - `IBAction` to connect the touch event to a method in your code
  - `UIStepper` to add an interface object that increases or decreases an integer
  - `UISegmentedControl` to control the number of `UIStepper` segments and the text or image of those segments
  - `UISwitchObject` to add a user interface object that is either on or off
- To enable users to save and retrieve data in your app, you can do the following:
  - Implement the `NSCoding` protocol before saving data
  - Use the `standardUserDefaults` instance to save data
  - Use the `objectForKey:` method and the same `bandObjectKey` to retrieve an object from `standardUserDefaults`
  - Use a `UIActionSheet` to delete saved data