11 Handling User Input

In Lesson 9 you were introduced to the `UIButton` and `UILabel` classes. The `UILabel` class enables you to display static text on the screen. In this lesson, you learn to use text fields and text views to accept input from users. Text fields enable users to type a single line of text and are instances of the `UITextField` class. Text views, on the other hand, enable users to type in multiple lines of text and are instances of the `UITextView` class. Both classes are part of the UIKit framework.

TEXT FIELDS

To create a text field, simply drag and drop a Text Field object from the Object library onto a storyboard scene (see Figure 11-1).

You can use the Attribute inspector to set up several attributes of the text field, including the Placeholder, Alignment, Border Style, Text Color, Font, and the type of keyboard that is displayed when the user taps on the text field (see Figure 11-2).

A *placeholder* is some text that is displayed in the text field when it is empty, typically prompting the user to enter some information in the field. You can choose from seven different keyboards to associate with a text field; the choice you make will depend on the type of data you expect. These keyboard styles can be selected using the Attribute inspector and are displayed in Figure 11-3.

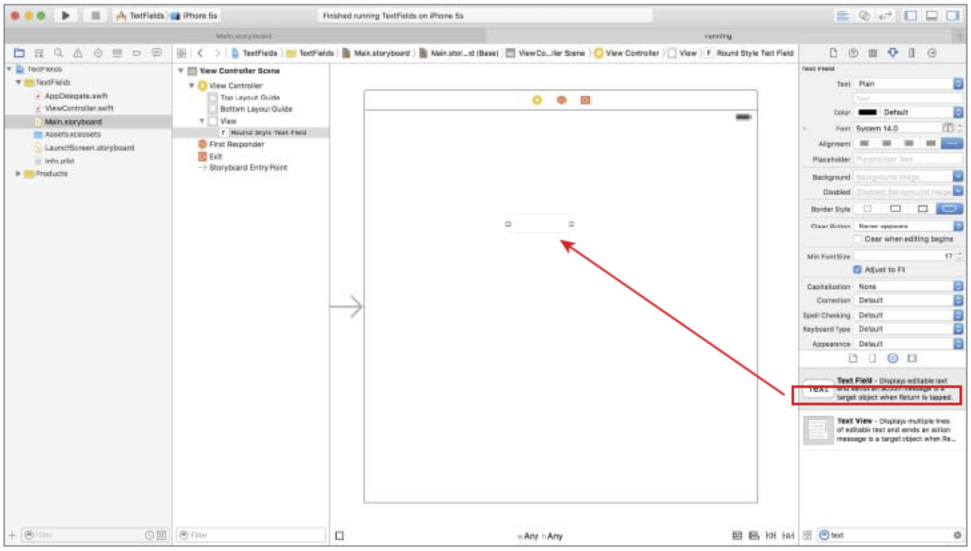
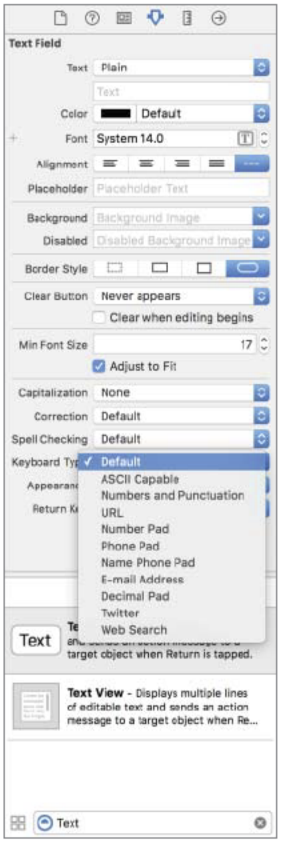

Text Field - Displays editable text and sends an action message to a target object when Return is tapped.
Text View - Displays multiple lines of editable text and sends an action message to a target object when Re...

FIGURE 11-1


Text Field

Text: Plain
Color: Default
Font: System 14.0
Placeholder: Placeholder Text
Background: Background Image
Disabled: Disabled Background Image
Clear Button: Never appears
Clear when editing begins
Min Font Size: 17
Adjust to Fit
Capitalization: None
Correction: Default
Spell Checking: Default
Keyboard Type: ✓ Default

Default
ASCII Capable
Numbers and Punctuation
URL
Number Pad
Phone Pad
Name Phone Pad
E-mail Address
Decimal Pad
Twitter
Web Search
Text - Text Field - Displays editable text and sends an action message to a target object when Return is tapped.
Text View - Displays multiple lines of editable text and sends an action message to a target object when Re...
Text

FIGURE 11-2

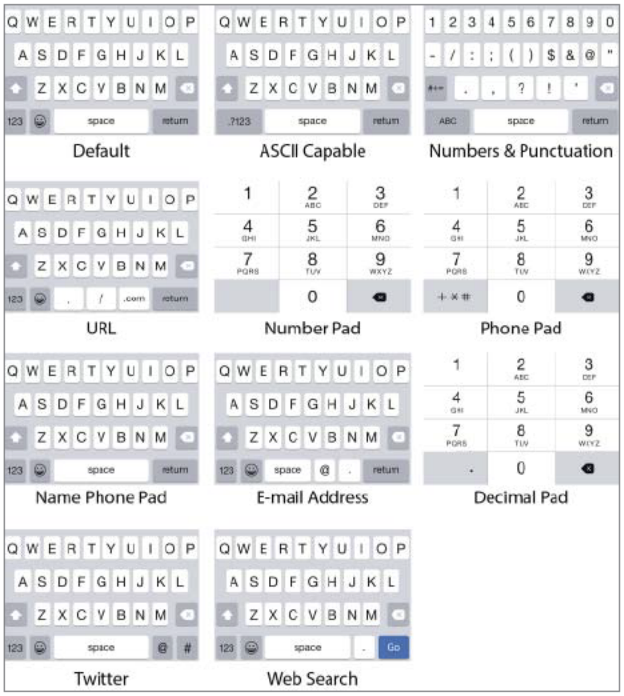


FIGURE 11-3

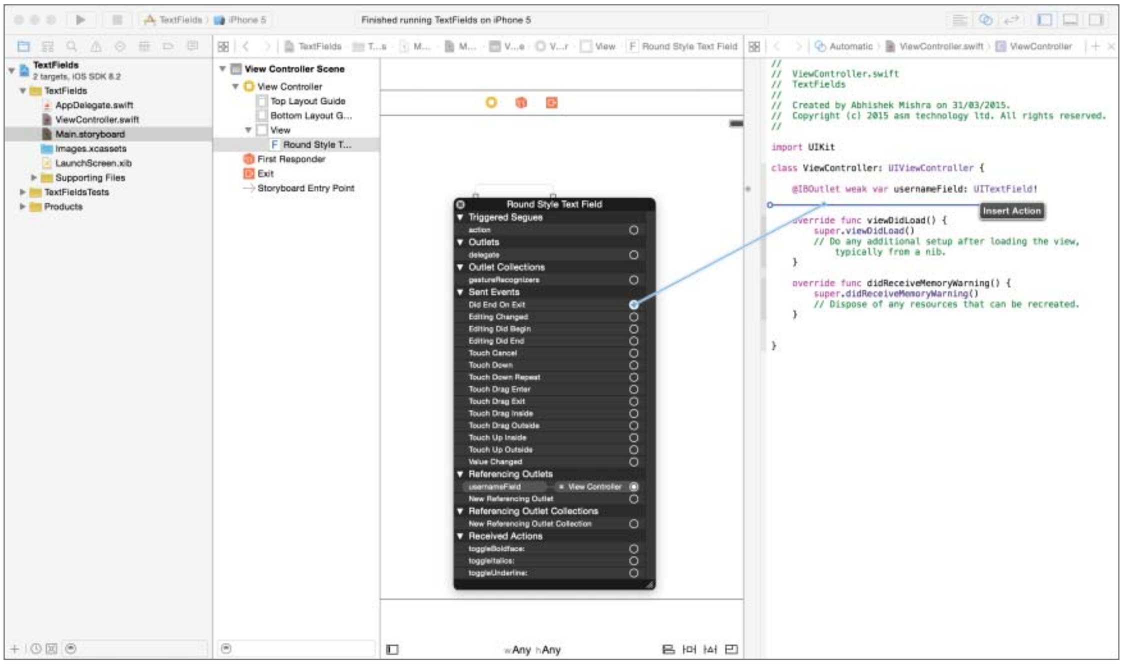
The text displayed in a text field is an instance of a `String` object. The `String` class is defined in the Foundation framework, and its instances represent sequences of characters (alphabets, numbers, punctuation marks).

To be able to access the text displayed in a text field object from code, you first need to create an outlet in the view controller class and then read the value of the `text` property in your code. For example, if `usernameField` is an outlet created using the assistant editor, you can use the following code to get the text displayed in the field:

```
let text:String = usernameField.text;
```

Tapping on a text field signifies that the user wants to interact with it, and as a result makes it the *active user interface element*. The active user interface element is formally known as the "first responder." When a text field receives first responder status, it automatically displays a keyboard.

To dismiss a keyboard when the Done button is pressed on the keypad, you will have to use the assistant editor to create a method in the view controller class and connect it to the `Did End On Exit` event of the text field (see Figure 11-4). A method in a view controller class that is wired to one of the events generated by a user interface element is commonly referred to as an action method.



```
//
//  ViewController.swift
//  TextFields
//
//  Created by Abhishek Mishra on 31/03/2015.
//  Copyright (c) 2015 asm technology ltd. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var usernameField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
           typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

FIGURE 11-4

In your action method, you need to ask the text field to resign from first responder status. You can do this by calling the `resignFirstResponder` method of the text field object as shown in the following snippet:

```
@IBAction func onDismissKeyboard(sender: AnyObject) {
        self.usernameField.resignFirstResponder();
    }
```

Note that the `sender` parameter will contain a reference to the source of the event that triggered this method (which will be the text field).

This method of dismissing the keypad works for most keyboard styles, except for the numeric keypads, which don't have a Done button. It is common practice for applications to allow the user to tap the background of the screen (outside the keypad or any other text field) to dismiss the keypad. One way to achieve this is by using a `UITapGestureRecognizer` object. Gesture recognizers are covered in detail in Lesson 21. For the moment, you can add a gesture recognizer to the view controller class by following these simple steps.

1. Add the following method declaration to the view controller class:

```
func handleBackgroundTap(sender: UITapGestureRecognizer) {

    }
```

2. Add the following code to the `viewDidLoad` method of the view controller class:

```
let tapRecognizer = UITapGestureRecognizer(target:self ,
              action: Selector("handleBackgroundTap:"))

tapRecognizer.cancelsTouchesInView = false
self.view.addGestureRecognizer(tapRecognizer)
```

3. Implement the `handleBackgroundTap:` method as follows:

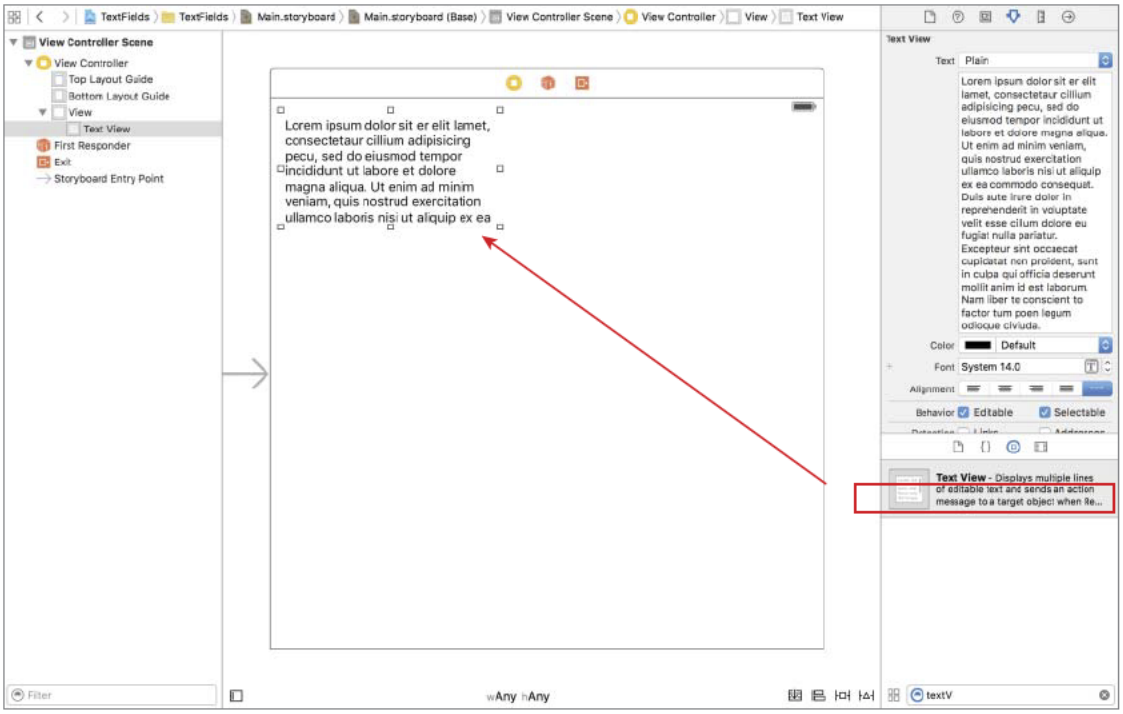
```
func handleBackgroundTap(sender: UITapGestureRecognizer) {
        self.usernameField.resignFirstResponder();
    }
```

TEXT VIEWS

Text views are similar to text fields in many respects. The key difference, however, is that text views can handle multiple lines of text. Text views handle the scrolling of text automatically, and can also be used as a read-only view, thus providing a convenient way to display scrollable multi-line text.

To create a text view, simply drag and drop a Text View element from the Object library onto the view (see Figure 11-5). By default a text view is sized to fit the entire screen, but you can resize/reposition it as needed.

To create a read-only text view, simply uncheck its Editable property in the Attribute inspector. A read-only text view does not display a keypad when tapped. Editable text views also enable you to select from one of seven different keypad types that will appear when the user taps them. The keypad associated with a text view, however, does not have a Done button; instead, it has a Return button that adds a new line to the text. Thus, to dismiss the keypad you will have to use the gesture recognizer technique discussed for text fields.



Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea

Text View

Text: Plain

Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cilum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Nam liber te conscient to factor tum poen legum odioque civiuda.

Color: Default

Font: System 14.0

Alignment

Behavior: ✓ Editable ✓ Selectable

Text View - Displays multiple lines of editable text and sends an action message to a target object when Re...

FIGURE 11-5

TRY IT

In this Try It, you create a new Xcode project based on the Single View Application template called `LoginSample` that presents a simple user interface to collect a username and password combination from the user. The user interface will also contain a Login button that displays a customized greeting to the user when it is tapped.

Lesson Requirements

- Launch Xcode.
- Create a new project based on the Single View Application template.
- Edit the storyboard with Interface Builder.

- Add two `UILabel` instances to the default scene, with the text `User name:` and `Password:`, respectively.
- Add two `UITextField` instances to the same scene, corresponding to the username and password fields, and create appropriate outlets in the view controller for them.
- Create an action method called `dismissKeyboard()` in the view controller class that calls the `resignFirstResponder` method on each text field, and connect the Did End On Exit event of each text field to this action method.
- Add a `UIButton` instance to the scene that, when tapped, displays a message in an alert view.
- Use a tap gesture recognizer to dismiss the keyboard when the background is tapped.

REFERENCE *The code for this Try It is available at* `www.wrox.com/go/swiftios`.

Hints

- To show the Object library, use the View ➪ Utilities ➪ Show Object Library menu item.
- To show the assistant editor, use the View ➪ Assistant Editor ➪ Show Assistant Editor menu item.

Step-by-Step

- Create a Single View Application in Xcode called `LoginSample`.
  1. Launch Xcode and create a new application by selecting File ➪ New Project.
  2. Select the Single View Application template from the list of iOS project templates.
  3. In the project options screen, use the following values:
     - Product Name: LoginSample
     - Organization Name: your company
     - Organization Identifier: com.yourcompany
     - Language: Swift
     - Devices: iPhone
     - Use Core Data: Unchecked
     - Include Unit Tests: Unchecked
     - Include UI Tests: Unchecked
  4. Save the project to your hard disk.

- Open the `Main.storyboard` file in the Xcode editor.
  1. Ensure the project navigator is visible and the LoginSample project is open.
  2. Click the `Main.storyboard` file.
- Add two `UILabel` instances to the default scene.
  1. Ensure the Object library is visible. To show it, select View ➪ Utilities ➪ Show Object Library.
  2. From the Object library, drag and drop two Label objects onto the scene.
  3. Use the Attribute inspector to set the `text` attribute of the first label to `User name:`. To show the Attribute inspector, select View ➪ Utilities ➪ Show Attributes Inspector.
  4. Change the `text` attribute of the second label to `Password:`.
  5. Select both labels in the scene and select Editor ➪ Size to Fit Contents to ensure the labels are large enough to show their contents.
  6. Add the following constraints using the pin constraints dialog box for the user name label:
     - Ensure the Constrain to margins option is unchecked.
     - The distance from the left edge of the label to the view is 10.
     - The distance from the top of the label to the view is 15.
     - The width of the label is 91.
     - The height of the label is 21.
  7. Add the following constraints using the pin constraints dialog box for the password label:
     - Ensure the Constrain to margins option is unchecked.
     - The distance from left edge of the label to the view is 10.
     - The vertical distance between the two labels is 15.
     - The width of the label is 91.
     - The height of the label is 21.
  8. Update the frames to match the constraints you have set.
     - Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
     - Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.
- Add two `UITextField` instances to the scene.

1. From the Object library, drag and drop two Text Field objects onto the scene and position them beside the two labels created in the previous step.
2. Use the Attribute inspector to set the Placeholder attribute of the first text field to `Enter user name`.
3. Use the Attribute inspector to set the Placeholder attribute of the second text field to `Enter password`.
4. Select both text fields in the scene and select Editor ➪ Size to Fit Contents to ensure the labels are large enough to show their contents.
5. Select the user name field in the scene and click the Pin button to display the constraints editor. Set the following constraints.
    - ➤ Ensure that Constrain to margins is unchecked.
    - ➤ The distance between the text field and the label should be 15.
    - ➤ The distance from the top of the text field to the view should be 10.
    - ➤ The width of the text field should be 200.
    - ➤ The height of the text field should be 30.
6. Add the following constraints for the password field:
    - ➤ Ensure the Constrain to margins option is unchecked.
    - ➤ The distance between the text field and the label should be 15.
    - ➤ The vertical distance between the two text fields should be 10.
    - ➤ The width of the text field should be 200.
    - ➤ The height of the text fields should be 30.
7. Update the frames to match the constraints you have set.
    - ➤ Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
    - ➤ Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.

➤ Add a `UIButton` instance to the scene.

1. From the Object library, drag and drop a Button object onto the scene.
2. Double-click it and set the text in the button to `Login`.
3. Select the button in the scene and click the Pin button to display the constraints editor. Set the following constraints:

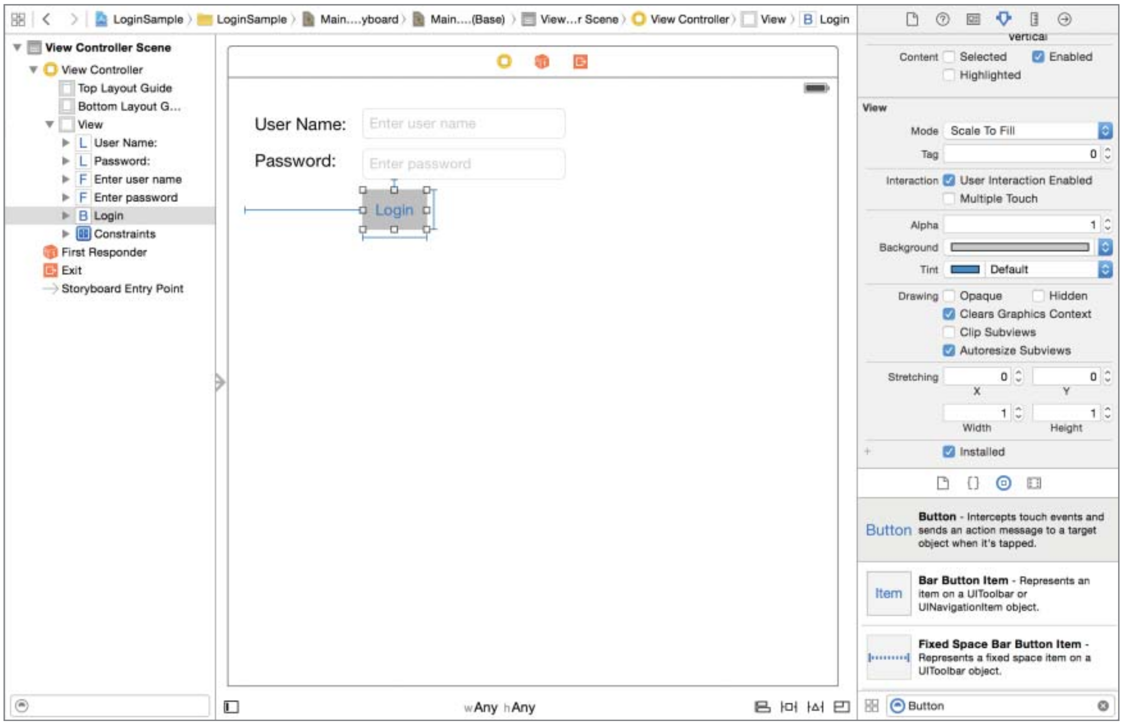
- ➤ Ensure the Constrain to margins option is unchecked.
- ➤ The horizontal distance between the button and the view should be 116.
- ➤ The vertical distance between the button and the password field should be 10.
- ➤ The width of the button should be 64.
- ➤ The height of the button should be 40.

4. Change the background color for the button to a dark gray color so that it is visible against a white background.

5. Update the frames to match the constraints you have set.

   - ➤ Click on the View controller item in the dock above the storyboard scene. This is the first of the three icons located directly above the selected storyboard scene.
   - ➤ Select Editor ➪ Resolve Auto Layout Issues ➪ Update Frames.

Your storyboard should resemble Figure 11-6.



Content: Selected ☐, Enabled ☑, Highlighted ☐

View
Mode: Scale To Fill
Tag: 0
Interaction: ☑ User Interaction Enabled, ☐ Multiple Touch
Alpha: 1
Background:
Tint: Default
Drawing: ☐ Opaque, ☐ Hidden, ☑ Clears Graphics Context, ☐ Clip Subviews, ☑ Autoresize Subviews
Stretching: X 0, Y 0, Width 1, Height 1
☑ Installed

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.

Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.

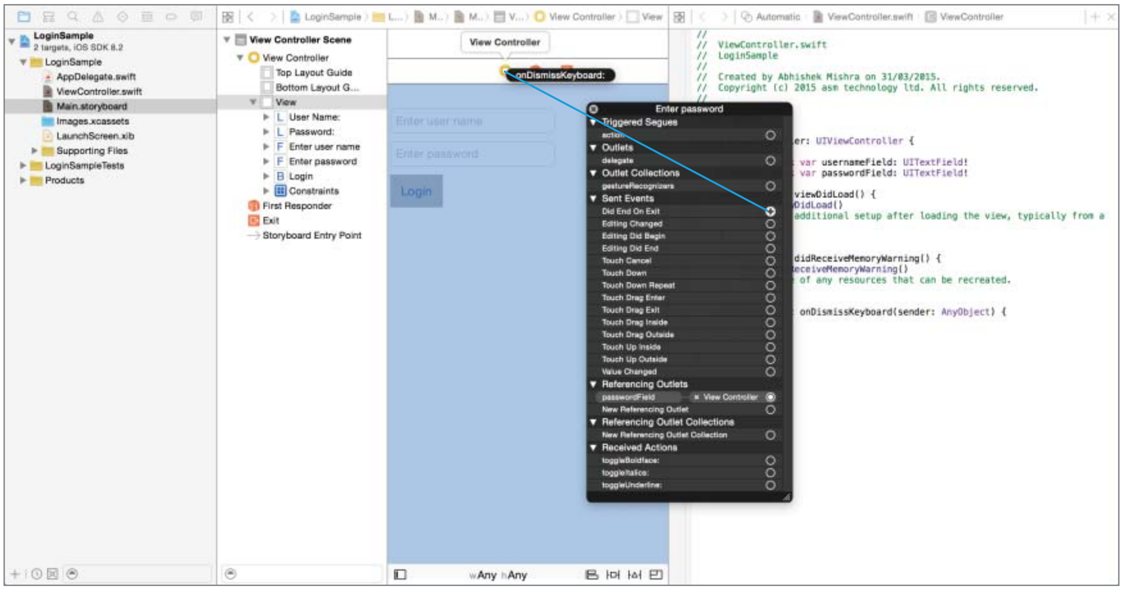
FIGURE 11-6

- Create outlets in the `ViewController` class and connect these outlets to the text fields in the scene.

  1. Ensure the assistant editor is visible. To show it, select View ➪ Editor ➪ Show Assistant Editor.
  2. Right-click the `UITextField` object corresponding to the user name to display a context menu. Drag from the circle beside the New Referencing Outlet option in the context menu to an empty line in the `ViewController.swift` file.
  3. Name the new outlet `usernameField`.
  4. Repeat this procedure for the password text field, and name the corresponding outlet `passwordField`.

- Create an action method in the `ViewController` class and associate it with the Did End On Exit events of the two text fields.

  1. Right-click the `UITextField` object corresponding to the username to display its context menu, and drag from the circle beside the Did End On Exit item to an empty line in the `ViewController.swift` file.
  2. Name the new Action `onDismissKeyboard`.
  3. Right-click the `UITextField` object corresponding to the password to display its context menu, and drag from the circle beside the Did End On Exit item to the icon representing the view controller in the dock (see Figure 11-7).



```
//
//  ViewController.swift
//  LoginSample
//
//  Created by Abhishek Mishra on 31/03/2015.
//  Copyright (c) 2015 asm technology ltd. All rights reserved.
//

er: UIViewController {

  var usernameField: UITextField!
  var passwordField: UITextField!

viewDidLoad() {
DidLoad()
additional setup after loading the view, typically from a

didReceiveMemoryWarning() {
eceiveMemoryWarning()
 of any resources that can be recreated.

 onDismissKeyboard(sender: AnyObject) {
```

FIGURE 11-7

4. Release the mouse button over the yellow view controller icon in the dock to present a list of existing action methods in the view controller. Select the `onDismissKeyboard` method.
5. Click the `ViewController.swift` file in the project navigator to open it.
6. Add the following code to the implementation of the `onDismissKeyboard` method:

```
usernameField.resignFirstResponder()
passwordField.resignFirstResponder()
```

➤ Create an action in the `ViewController.swift` file and connect it with the Touch Up Inside event of the login button.

1. Select the storyboard in the project navigator.
2. Right-click the Login button in the scene to display its context menu, and drag from the circle beside the Touch Up Inside item to an empty line in the `ViewController.swift` file.
3. Name the new action method `onLogin`.
4. Click the `ViewController.swift` file in the project navigator to open it.
5. Add the following code to the implementation of the `onLogin` method:

```
usernameField.resignFirstResponder()
passwordField.resignFirstResponder()

let userName:String = usernameField.text!
let length:Int = userName.characters.count

if length == 0 {
    return
}

let alert = UIAlertController(title: "",
    message: "Login succesfull",
    preferredStyle: UIAlertControllerStyle.Alert)

alert.addAction(UIAlertAction(title: "Ok",
    style: UIAlertActionStyle.Default,
    handler: nil))

self.presentViewController(alert,
    animated: true,
    completion: nil)
```

➤ Add a tap gesture recognizer and use it to dismiss the keyboard when the background area of the view is tapped.

1. Add the following code to the `viewDidLoad` method of the `ViewController.Swift` file, after the `super.viewDidLoad()` line:

```
let tapRecognizer = UITapGestureRecognizer(target:self ,
                         action: Selector("handleBackgroundTap:"))
```

```
tapRecognizer.cancelsTouchesInView = false
self.view.addGestureRecognizer(tapRecognizer)
```

2. Implement the `handleBackgroundTap()` method in the `ViewController.swift` file as follows:

```
func handleBackgroundTap(sender: UITapGestureRecognizer) {
        usernameField.resignFirstResponder()
        passwordField.resignFirstResponder()
    }
```

➤ Test your app in the iOS Simulator.

Click the Run button in the Xcode toolbar. Alternatively, you can use the Project ➪ Run menu item.