ГЛАВА 3

Основы взаимодействия

Наше приложение Hello World было хорошим введением в разработку приложений для системы iOS с помощью интегрированной среды Xcode и каркаса Cocoa Touch, но в нем не было очень важной функциональной возможности: взаимодействия с пользователем. Без этого приложение имеет очень ограниченное применение.

В этой главе мы напишем немного более сложное приложение, в котором будут две кнопки и метка, как показано на рис. 3.1. Когда пользователь нажмет одну из кнопок, текст метки изменится. Этот пример может показаться слишком упрощенным, но он демонстрирует ключевые концепции, связанные с реализацией взаимодействия пользователя с приложениями для системы iOS.

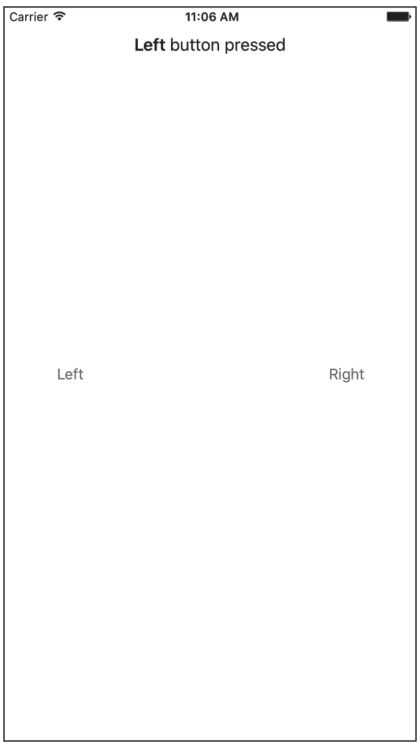

Рис. 3.1. Простое приложение с двумя кнопками, которое мы разработаем в этой главе

Парадигма “модель–контроллер–представление”

Концепция “**модель–контроллер–представление”** (Model-View-Controller — MVC) представляет собой очень логичный способ разделения кода, лежащего в основе приложений с графическим пользовательским интерфейсом. В настоящее время практически все объектно-ориентированные среды разработки в той или иной степени используют концепцию MVC, но лишь некоторые из них действительно полностью воплощают парадигму MVC, как это делает среда Cocoa Touch.

Шаблон MVC разделяется по функциональным возможностям на три категории.

- Модель. Состоит из классов, в которых хранятся данные приложения.
- Представление. Создает окна, элементы управления и другие элементы, которые пользователь видит и с которыми взаимодействует.
- Контроллер. Связывает модель и представление, реализует логику приложения, в соответствии с которой оно обрабатывает данные, введенные пользователем.

Назначение концепции MVC — создать как можно более независимые один от другого объекты, реализующие эти три типа кода. Любой объект, создаваемый вами, должен четко идентифицироваться как объект, принадлежащий одной из перечисленных выше категорий. При этом он должен вообще не иметь или иметь как можно меньше функциональных возможностей, которые можно было бы отнести к остальным двум категориям. Например, объект, реализующий кнопку, не должен содержать код для обработки данных в момент ее нажатия, а реализация банковского счета не должна содержать код для рисования таблицы для демонстрации транзакций.

Концепция MVC обеспечивает максимальное повторное использование кода. Класс, реализующий обобщенную кнопку, можно использовать в любом приложении. Класс, реализующий кнопку, выполняющую конкретные вычисления при ее нажатии, можно использовать только в том приложении, для которого он был написан изначально.

Когда вы пишете приложения в среде Cocoa Touch, вы в основном создаете компоненты представления, используя визуальный редактор Interface Builder, хотя иногда вы также модифицируете свой интерфейс с помощью кода или создаете подклассы для существующих видимых деталей и элементов управления.

Ваша модель будет создана на основе классов языка Swift, разработанных для хранения данных приложения. Мы не собираемся создавать объекты модели в этой главе, поскольку не планируем хранить или собирать данные, и описываем их для того, чтобы использовать впоследствии при разработке более сложных приложений.

Ваш контроллер будет состоять из классов, создаваемых вами, а также относящихся к вашему приложению. Контроллер может полностью состоять из обычных классов, но чаще они являются подклассами одного из существующих обобщенных классов контроллера из библиотеки UIKit, например класса `UIViewController`, с которым мы встретимся в следующем разделе. Создавая подклассы одного из существующих классов, вы получаете в свое полное распоряжение множество функциональных возможностей и экономите время за счет того, что не изобретаете велосипед.

По мере углубления в среду Cocoa Touch вы быстро убедитесь, что классы каркаса UIKit следуют принципам MVC. Если вы будете последовательно придерживаться этой концепции в процессе разработки, то в результате создадите более ясный и легко эксплуатируемый код.

Создание приложения ButtonFun

Настало время создать следующий проект Xcode. Мы собираемся использовать тот же шаблон, который исследовали в предыдущей главе: `Single View Application`. Отталкиваясь от этого простого шаблона, нам будет легче увидеть, как взаимодействуют объекты представления и шаблона в рамках приложения для системы iOS. В следующих главах мы будем использовать другие шаблоны.

Запустите программу Xcode и выберите команду File⇨New⇨New Project... или нажмите комбинацию клавиш <⌘+N>. Выберите шаблон Single View Application и щелкните на кнопке Next.

Вы увидите тот же самый лист настроек, который видели в предыдущей главе. В поле Product Name введите название нового приложения — `ButtonFun`. Поля Organization Name, Company Identifier и Language идентификатора пакета должно содержать те же значения, которые мы использовали в предыдущей главе, поэтому трогать его не следует. Мы планируем использовать механизм Auto Layout, чтобы наше приложение работало на любых устройствах iOS, поэтому в поле Devises выберите значение Universal. Полый список настроек представлен на рис. 3.2.

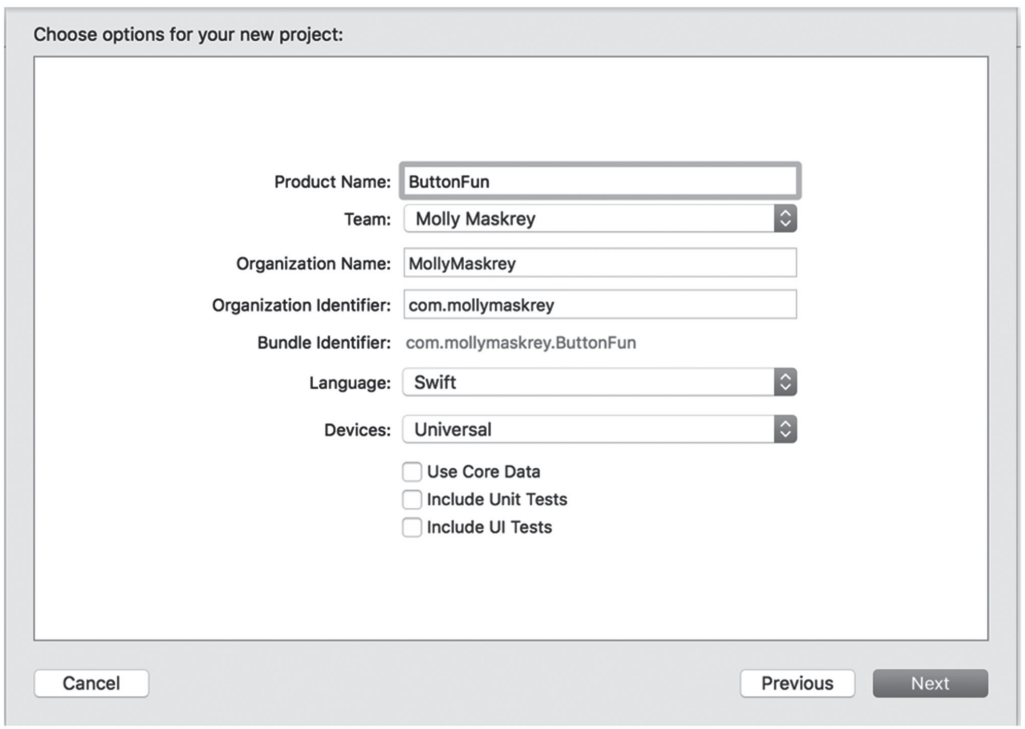

Рис. 3.2. Выбор имени проекта и его параметров

Щелкните на кнопке Next и выберите место хранения своего проекта. Оставьте флажок Create Git repository сброшенным или установленным на свое усмотрение. Сохраните проект среди остальных проектов нашей книги.

Создание контроллера представления

Немного позднее мы разработаем представление (т.е. пользовательский интерфейс) для нашего приложения, используя программу Interface Builder, как это было сделано в предыдущей главе. А пока мы собираемся внести изменения в файлы исходного кода, созданного для нас шаблоном проекта. Да, мы действительно собираемся написать часть кода в этой главе. Но прежде чем вносить какие-либо изменения, взглянем на файлы, сгенерированные шаблоном. Для этого следует раскрыть навигатор проекта, в котором уже раскрыта группа Button Fun. Если это не так, ее следует открыть, щелкнув на треугольнике раскрытия, расположенном рядом с ее именем (рис. 3.3).

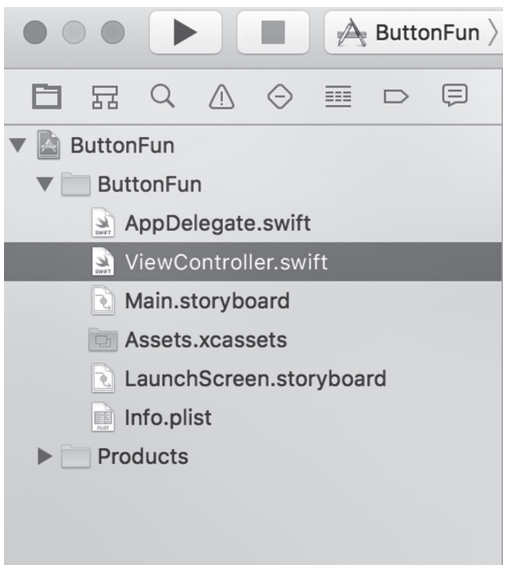

Рис. 3.3. Навигатор проекта, демонстрирующий файлы классов, созданные шаблоном проекта. Обратите внимание на то, что наш префикс класса автоматически инкорпорирован в имена класса

Группа `Button Fun` должна содержать два файла с исходным кодом, один файл раскадровки, файл заставки, каталог ресурсов, содержащий все изображения, необходимые для приложения, и файл `Info.plist`, который будет рассмотрен в последующих главах. Два файла исходных кодов содержат реализации классов, необходимых для приложения: делегат приложения и контроллер представления приложения, имеющего только одно представление. Делегат

приложения будет рассмотрен позднее в этой главе, а пока исследуем контроллер создаваемого нами представления.

Класс контроллера, ответственный за управление этим представлением, называется `ViewController`. Это имя означает, что класс является контроллером представления. Щелкните на файле View.Controller.swift в окне навигатора проекта, и вы увидите на экране содержимое этого файла (листинг 3.1).

Листинг 3.1. Код класса `ViewController`, сгенерированного по шаблону

```
import UIKit

class ViewController: UIViewController {

   override func viewDidLoad() {
      super.viewDidLoad()
      // Дополнительные настройки после загрузки представления,
      // как правило, из nib-файла.
}

    override func didReceiveMemoryWarning() {
       super.didReceiveMemoryWarning()
       // Освобождаем любые восстанавливаемые ресурсы
    }
}
```

Поскольку этот файл сгенерирован по шаблону, он имеет не очень богатое содержание. `ViewController` — это подкласс класса `UIViewController`, одного из обобщенных классов контроллера, которые мы упоминали выше. Он является частью библиотеки UIKit и предоставляет в наше распоряжение множество функциональных возможностей. Среда Xcode не знает, какие именно функциональные возможности должно иметь наше приложение, но ей известно, что мы собираемся сделать нечто, поэтому она создала этот класс, в котором будут реализованы требуемые функциональные возможности.

Выходы и действия

В главе 2 для разработки пользовательского интерфейса мы использовали программу Interface Builder, а в листинге 3.1 мы видели оболочку класса контроллера. Рассмотрим способ, с помощью которого наш код в классе контроллера представления будет правильно взаимодействовать с объектами (кнопками, метками и пр.), содержащимися в раскадровке. Класс контроллера может ссылаться на объекты в раскадровке, используя особый вид свойства под названием выход (outlet). Выход можно интерпретировать как указатель, ссылающийся на объект в пользовательском интерфейсе. Например, допустим, что вы создали текстовую метку с помощью программы Interface Builder (как это было сделано в главе 2) и хотите изменить ее текст, модифицируя исходный код. Объявив выход и связав его с объектом метки, вы можете использовать эту переменную в своем коде для изменения текста, изображенного на метке. Мы покажем, как это сделать, ниже в этой главе.

Перейдем на противоположную сторону. Объекты интерфейса в нашей раскадровке или nib-файле могут быть связаны со специальными методами в классе контроллера. Эти специальные методы называются действиями (actions). Вы даже можете сообщить программе Interface Builder, что, когда пользователь касается кнопки, она должна вызвать один метод, а когда пользователь отнимает палец от кнопки, должен быть вызван другой метод действия.

Среда Xcode поддерживает разные способы создания выходов и действий. Например, можно указать их в заголовочном файле контроллера представления и лишь после этого открывать программу Interface Builder и приступать к связыванию выходов и действий, однако представление помощника (assistant view) в программе Xcode позволяет быстрее и проще создавать и связывать выходы и действия одновременно. Этот процесс мы также вскоре рассмотрим. Но прежде чем приступать к установке связей между выходами и действиями, поговорим о них немного подробнее. Выходы и действия — два основных элемента, используемых при создании приложений для операционной системы iOS, поэтому нам важно понимать, что они собой представляют и как работают.

Выходы

Выход (outlet) — это обычное свойство в языке Swift, объявляемое с помощью атрибута `@IBOutlet`. Объявление выхода в заголовочном файле контроллера должно выглядеть примерно следующим образом:

```
@IBOutlet weak var myButton: UIButton!
```

Этот пример демонстрирует выход с именем `myButton`, который можно связать с любой кнопкой пользовательского интерфейса.

Видя атрибут `@IBOutlet`, компилятор Swift не делает ничего особенного. Его единственное назначение — подсказать программе Interface Builder, что это свойство, которое будет связано с объектом в раскадровке или nib-файле. Любом свойству, которое вы создадите и захотите связать с объектом в nib-файле, должен предшествовать атрибут `@IBOutlet`. К счастью, программа Xcode теперь создает выходы автоматически при перетаскивании объекта на свойство, с которым вы хотите его связать, и даже при перетаскивании его на класс, в котором вы хотите создать новый выход.

У читателей может возникнуть вопрос, почему объявление свойства `myButton` завершается восклицательным знаком. По правилам языка Swift все свойства должны быть полностью инициализированы до завершения работы всех инициализаторов, если они не были объявлены как свойства необязательного типа. Когда контроллер загружает раскадровку, значения свойств выходов определяются информацией, содержащейся в раскадровке, но это происходит *после* запуска инициализатора контроллера представления. В результате, если явным образом не определить фиктивные значения (что нежелательно), то свойства выхода необходимо объявить как свойства необязательного типа. Таким образом, существует

два способа объявления свойств выходов — с помощью символов ! и ?, как показано в листинге 3.2.

Листинг 3.2. Два способа объявления переменных необязательного типа

```
@IBOutlet weak var myButton1: UIButton?
@IBOutlet weak var myButton2: UIButton!
```

Второй способ проще первого, потому что не требует явной распаковки переменной необязательного типа, когда она впоследствии понадобится в коде контроллера представления (листинг 3.3).

Листинг 3.3. Исключение необходимости явно распаковывать переменные необязательного типа

```
let button1 = myButton1!  // Переменную необязательного типа
                          // необходимо явно распаковать
let button2 = myButton2   // Переменная myButton2 распакована неявно
```

ЗАМЕЧАНИЕ. Спецификатор `weak` перед объявлением свойства выхода означает, что данное свойство не обязано создавать сильную ссылку на кнопку. Объекты автоматически удаляются из памяти, если на них больше нет сильных ссылок. В данном случае нет никакого риска, что кнопка будет удалена из памяти, ведь на нее всегда будет указывать сильная ссылка, поскольку кнопка является частью пользовательского интерфейса. Объявление слабой ссылки позволяет удалять представление из памяти, если оно больше не нужно, одновременно исключая его из пользовательского интерфейса. При этом ссылка устанавливается равной `nil`.

Действия

Действия (actions) — это методы, возвращающие объекты с атрибутом `@IBAction`, которые сообщают программе Interface Builder, что данный метод может быть активизирован элементом управления в раскадровке или nib-файле. Как правило, объявление действия выглядит примерно так:

```
@IBAction func doSomething(sender: UIButton) {}
```

Или так:

```
@IBAction func doSomething() {}
```

Реальное имя метода может быть любым. Обычно действие либо не имеет аргументов, либо получает один аргумент, который, как правило, имеет имя `sender`. При вызове метода действия аргумент `sender` содержит ссылку на вызвавший его объект. Таким образом, например, если действие было вызвано в результате нажатия кнопки, аргумент `sender` содержит ссылку на конкретную кнопку, на которой произошло нажатие. Благодаря аргументу `sender` существует возможность отвечать нескольким элементам управления, используя один

и тот же метод действия. Он позволяет идентифицировать элемент управления, вызвавший метод действия.

СОВЕТ. На самом деле существует третье, редко используемое объявление метода действия, которое выглядит так:

```
@IBAction func doSomething(sender: UIButton,
                forEvent event: UIEvent){};
```

Этот вид объявления целесообразно использовать, если вам нужна дополнительная информация о событии, порожденном вызванным методом. Управляющие события рассматриваются в следующей главе.

Нет ничего плохого в том, что мы объявили метод действия с аргументом `sender`, а потом проигнорировали его. Методы действия в каркасе Cocoa и системе NeXTSTEP должны получать аргумент `sender` независимо от того, используют они его или нет, поэтому многие программы для системы iOS написаны именно так.

Разобравшись в том, что такое действия и выходы, перейдем к их применению при разработке пользовательского интерфейса. Однако, прежде чем начать, необходимо уделить немного времени вопросам, связанным с наведением порядка.

Разработка контроллера представления

Щелкните на файле ViewController.swift в навигаторе проекта, чтобы открыть файл реализации. Как видим, выбранный нами шаблон проекта сгенерировал совсем немного шаблонного кода в виде методов `viewDidload()` и `didReceiveMemoryWarning()`. Эти методы обычно используются в подклассах класса `UIViewController`, поэтому программа Xcode предоставила нам их шаблонную реализацию и мы должны просто добавить сюда свой код. Однако большая часть этих шаблонных реализаций для нашего проекта не нужна, так что они лишь увеличивают размер файла и затрудняют чтение. Для того чтобы привести реализацию в порядок, удалим лишние фрагменты. В результате содержание файла должно стать таким, как показано в листинге 3.4.

Листинг 3.4. Упрощенный файл `ViewController.swift`

```
import UIKit

  class ViewController: UIViewController {

}
```

Разработка пользовательского интерфейса

Сохраните внесенные изменения, а затем щелкните на пункте Main.storyboard, чтобы открыть представление вашего приложения в окне Interface Builder (рис. 3.4). Как было указано в предыдущей главе, белое окно, открывшееся в области редактирования, является единственным представлением нашего приложения. Вернувшись к рис. 3.1, легко убедиться, что мы должны добавить в это представление две кнопки и одну метку.

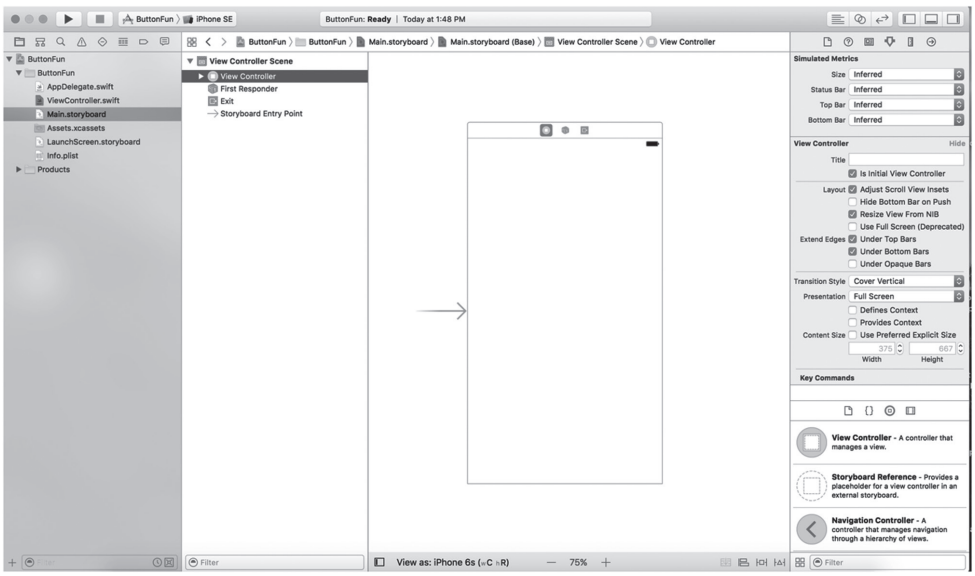

Рис. 3.4. Открытие файла `Main.storyboard` в программе Interface Builder среды Xcode

Подумаем секунду о нашем приложении. Мы собираемся добавить в интерфейс две кнопки и одну метку. Этот процесс очень похож на то, что мы делали в предыдущей главе. Однако теперь мы хотим использовать выходы и действия, чтобы наше приложение стало интерактивным.

Кнопки должны запускать методы действия нашего контроллера. Мы могли бы сделать так, чтобы вызывались разные методы действия, но, поскольку они будут выполнять, по существу, одно и то же задание (обновлять текст метки), нам необходимо вызывать один и тот же метод. Мы будем различать кнопки с помощью аргумента `sender`, который рассмотрели выше. Кроме метода действия, нам также нужен выход, связанный с меткой, чтобы мы могли изменять текст, отображаемый на метке.

Сначала добавим кнопки, а затем разместим метку. Разработав интерфейс, добавим к нему соответствующие действия и выходы. Кроме того, мы могли бы вручную объявить действия и выходы, а затем соединить их с элементами интерфейса, но среда Xcode сделает это автоматически.

Добавление кнопок и метода действия

Сначала добавим в интерфейс две кнопки. Затем среда Xcode создаст пустой шаблонный метод действия и мы свяжем с ним обе кнопки. После того как пользователь щелкнет на кнопке, будет вызван метод действия и выполнен любой код, который будет в нем записан.

Выберите команду View⇨Utilities⇨Show Object Library или нажмите комбинацию клавиш <Control+Option+⌘+3>, чтобы открыть библиотеку объектов. Введите в поле поиска библиотеки строку `UIButton`. На самом деле достаточно ввести только первые буквы `uibu`, чтобы сузить список (лучше вводить буквы в нижнем регистре, чтобы избежать путаницы при случайном нажатии клавиши <Shift>). После ввода этой строки в окне библиотеки объектов должен появиться только один объект: Button (рис. 3.5).

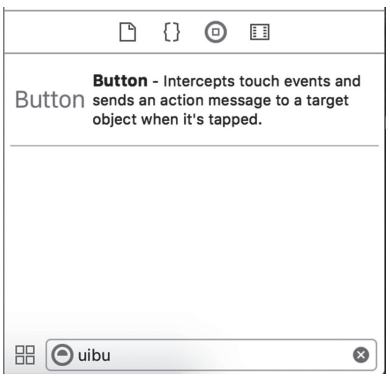

Рис. 3.5. В окне библиотеки объектов появляется элемент Button

Перетащите объект Button из библиотеки и оставьте его в белом окне области редактирования, чтобы добавить кнопку в представление нашего приложения. Разместите эту кнопку возле левого края представления на достаточном расстоянии, используя вертикальную голубую линию разметки. Для того чтобы выровнять кнопку по высоте, разместив ее посередине представления, используйте горизонтальную голубую линию разметки. Если это вам поможет, ориентируйтесь на рис. 3.1.

ЗАМЕЧАНИЕ. Голубые линии разметки, которые появляются при перемещении объектов в окне программы Interface Builder, помогут вам освоить принципы руководства iOS Human Interface Guidelines (HIG). Компания Apple разработала руководство HIG для людей, проектирующих приложения для устройств iPhone и iPad. Руководство HIG регламентирует, как следует (и не следует) проектировать пользовательский интерфейс. Вам необходимо прочитать его, потому что оно содержит ценную информацию, которую должен знать каждый разработчик приложений для устройства iPhone. Это руководство можно найти на веб-странице `https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/`.

Дважды щелкните на добавленной кнопке. Это даст возможность отредактировать ее название. Назовем эту кнопку Left.

Выберите команду View⇨Assistant Editor⇨Show Assistant Editor или нажмите комбинацию клавиш <Option+⌘+Return>. Вы можете показывать и скрывать помощник редактора, щелкая средней кнопкой в группе кнопок Editor, входящей в коллекцию из семи кнопок, расположенных в правом верхнем углу окна проектирования (рис. 3.6).

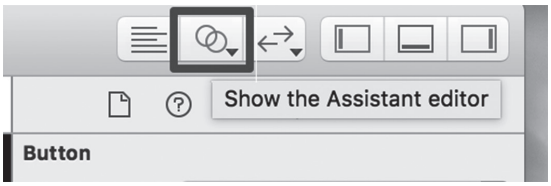

Рис. 3.6. Кнопка Show the Assistant Editor (две окружности)

Помощник редактора появляется в правой части окна редактирования. В окне редактирования помощник редактора автоматически открывает файл `ViewController.swift`, являющийся файлом реализации контроллера представления, владеющего представлением, которое вы видите на экране.

ПОДСКАЗКА. После открытия помощника редактора вам, возможно, понадобится изменить размеры окна, чтобы было достаточно места для работы. Если хотите работать с маленьким экраном, как, например, на компьютере MacBook Air, возможно, придется закрыть вспомогательное представление и/или навигатор проекта, чтобы эффективно работать с помощником редактора (рис. 3.8). Это легко сделать с помощью трех кнопок, расположенных в правом верхнем углу окна проекта (см. рис. 3.6).

Среда Xcode знает, что наш контроллер представления отвечает за демонстрацию представления в раскадровке, поэтому помощник редактора знает, что должен показать нам реализацию класса контроллера представления, в котором, скорее всего, будет происходить связывание выходов и действий. Однако, если все же вы не видите файла, который вам нужен, можете перейти на панель быстрого перехода в верхней части окна помощника редактора и устранить проблему. Сначала найдите сегмент панели быстрого перехода `Automatic` и щелкните на нем. На экране появится всплывающее меню, в котором необходимо выбрать команду Manual⇨Button Fun⇨ViewController.swift. Теперь на экране должен появиться правильный файл.

Попросим программу Xcode автоматически создать новый метод действия для нас и связать его с только что созданной кнопкой. Мы добавим эти определения в расширение класса контроллера представления. Для этого щелкните на кнопке, добавленной в раскадровку, чтобы она оказалась выбранной. Нажмите и удерживайте клавишу <Control>, а затем щелкните мышью и перетащите курсор с кнопки на окно помощника редактора. Вы увидите голубую линию,

которая будет следовать за курсором (рис. 3.8). Эта линия связывает объекты с кодом или другими объектами. Когда вы будете перемещать курсор в пределах определения класса, как показано на рис.3.8, на экране появится всплывающее меню, которое будет информировать вас, что отпускание кнопки мыши приведет к вставке выхода, действия или коллекции выходов.

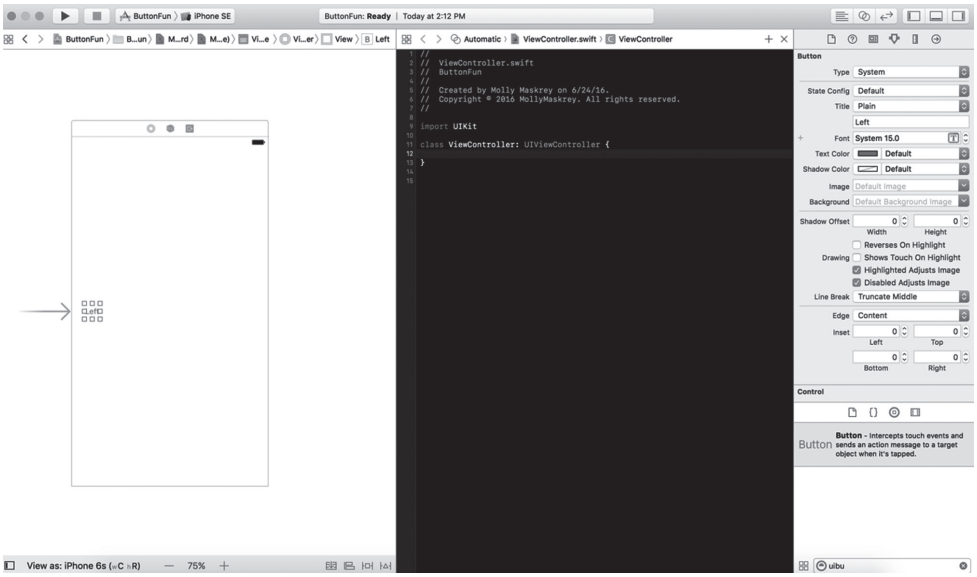

Рис. 3.7. Для того чтобы видеть окна редактирования на небольших дисплеях, возможно, придется закрыть другие представления

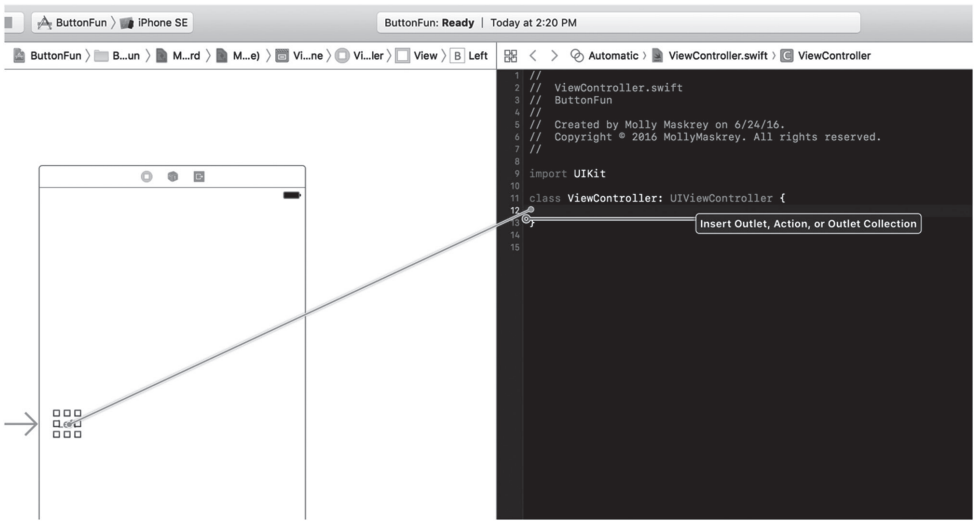

Рис. 3.8. Перетаскивание курсора в исходный код при нажатой клавише <Control> дает возможность создать выход, действие или коллекцию выходов

ЗАМЕЧАНИЕ. В настоящей книге мы используем действия и выходы, но не коллекции выходов. Коллекции выходов позволяют соединять несколько однородных объектов с одним и тем же свойством `NSArray`, а не создавать отдельное свойство для каждого объекта.

Для того чтобы закончить соединение, отпустите кнопку мыши, и на экране появится всплывающее меню (рис. 3.9).

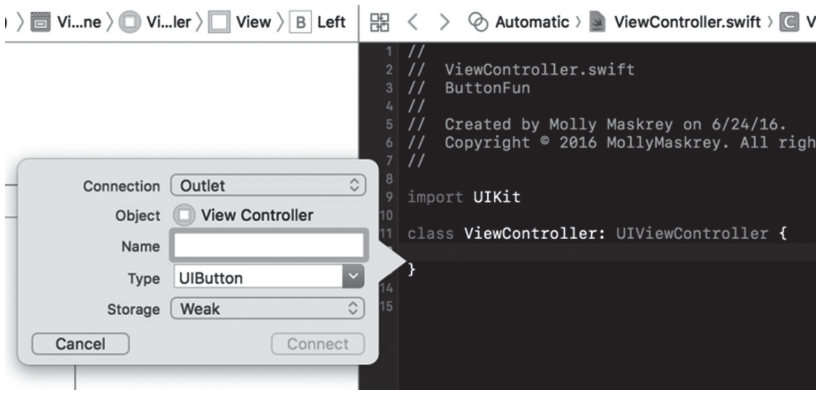

Рис. 3.9. Всплывающее меню, которое появляется после перетаскивания курсора в исходный код при нажатой клавише <Control>

Это окно позволяет настроить новое действие. Щелкните на меню Connection и измените выбор команды с Outlet на Action. Тем самым вы сообщите программе Xcode, что хотите создать действие, а не выход. В результате окно примет вид, показанный на рис. 3.10. Введите в поле Name строку `buttonPressed`. Когда закончите ввод, *не* нажимайте клавишу <Return>. Нажатие клавиши <Return> приведет к завершению процесса создания выхода, а мы пока не собираемся этого делать. Вместо этого нажмите клавишу <Tab>, перейдите в поле Type и введите в нем строку `UIButton`, заменив значение `AnyObject`, заданное по умолчанию.

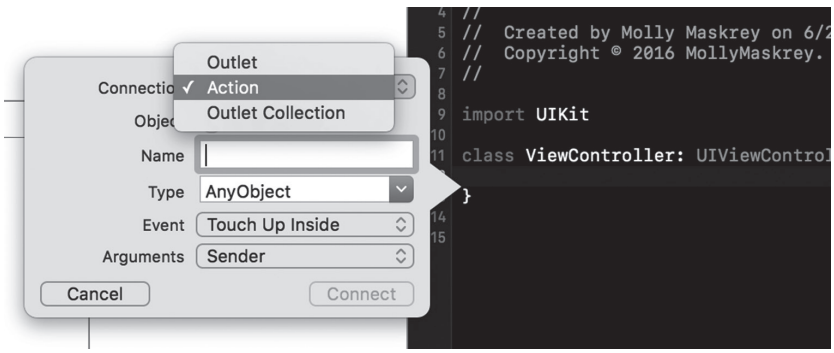

Рис. 3.10. Изменение типа соединения с Outlet на Action

Ниже поля Type расположены два поля, которые мы оставим заполненными значениями, заданными по умолчанию. Поле Event позволяет указать, когда будет вызван метод. Значение по умолчанию `Touch Up Inside` означает событие, когда пользователь отнимает палец от экрана над кнопкой. Это стандартное событие для кнопок. Оно дает пользователю возможность передумать. Если, перед тем как поднять палец, пользователь переместит его за пределы кнопки, метод не будет вызван.

Поле Arguments позволяет выбрать одну из трех разных сигнатур, используемых для методов действия. Мы выбрали аргумент `sender`, так что можем указать, какая кнопка вызвала метод. Это значение задается по умолчанию, поэтому оставим его неизменным.

Нажмите клавишу <Return> или щелкните на кнопке Connect, и программа Xcode вставит метод действия. Для файла `ViewController.swift` в окне помощника редактора это будет выглядеть так, как показано в листинге 3.5.

Листинг 3.5. Файл `ViewController.swift` с добавленным действием `IBAction`

```
import UIKit

class ViewController: UIViewController {

   @IBAction func buttonPressed(_ sender: UIButton) {
   }
}
```

Программа Xcode не только добавила объявление метода в код, но и связала кнопку с этим методом, сохранив эту информацию в раскадровке. Это значит, что нам не надо делать что-либо еще, чтобы кнопка вызвала этот метод при выполнении приложения.

Вернитесь к файлу `Main.storyboard` и перетащите на его окно другую кнопку, на этот раз поместив кнопку в правой части экрана. На экране появятся голубые линии, помогающие сориентировать кнопку по отношению к правому краю и другой кнопке. Поместив ее в требуемое место, дважды щелкните на ней и измените ее имя на Right.

ЗАМЕЧАНИЕ. Вместо перетаскивания нового объекта из библиотеки можно нажать клавишу <Option> и перетащить на представление оригинальный объект (в данном примере это кнопка `Left`). Удерживание кнопки <Option> заставляет программу Interface Builder скопировать перетаскиваемый объект.

Пока мы не хотим создавать новый метод действия. Вместо этого свяжем эту кнопку с существующим методом, созданным программой Xcode. Изменив имя кнопки, нажмите клавишу <Control>, щелкните на новой кнопке и снова перетащите ее на заголовочный файл. На этот раз, когда курсор достигнет объявления метода `buttonPressed()`, этот метод будет подсвечен

и на экране появится всплывающее окно Connect Action (рис. 3.11). Если вы не увидите это окно сразу, перетаскивайте указатель мыши по кругу до тех пор, пока оно не появится. Когда увидите это окно, отпустите кнопку мыши, и программа Xcode соединит эту кнопку с существующим методом действия. В результате при нажатии кнопки будет вызван тот же метод, что и для другой кнопки.

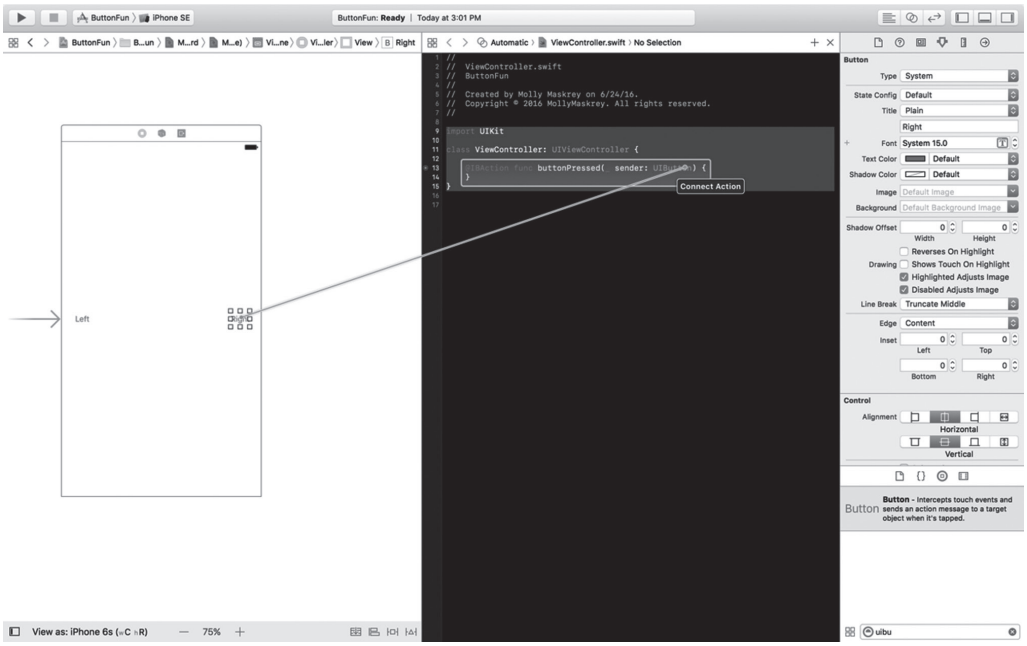

Рис. 3.11. Перетаскивая существующее действие, можно установить связь между ним и кнопкой

Добавление метки и выхода

Находясь в библиотеке объектов, введите в поле поиска слово `lab`, чтобы найти элемент интерфейса Label (рис. 3.12). Перетащите метку на свой пользовательский интерфейс и поместите где-нибудь между двумя кнопками. Затем, используя маркеры масштабирования, растяните метку от левого края (отмеченного голубой линией) до правого. Это обеспечит достаточно места для текста, который будет выведен для пользователя.

По умолчанию метки выравниваются по левому краю, но нашу метку мы хотим отцентровать. Выберите команду View⇨Utilities⇨Show Attributes Inspector (или нажмите комбинацию клавиш <Option+⌘+4>), чтобы открыть окно инспектора атрибутов (рис. 3.12). Выберите метку, а затем найдите в инспекторе атрибутов кнопки Alignment. Выберите среднюю кнопку Alignment, чтобы отцентровать текст метки.

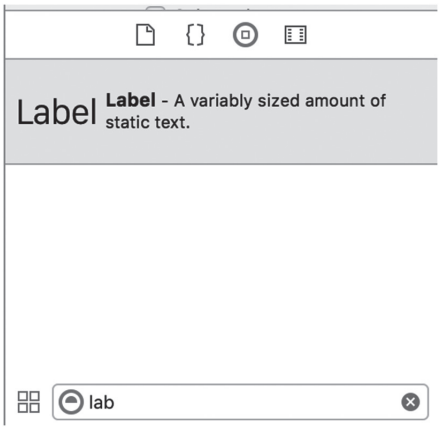

Рис. 3.12. Метка в библиотеке объектов

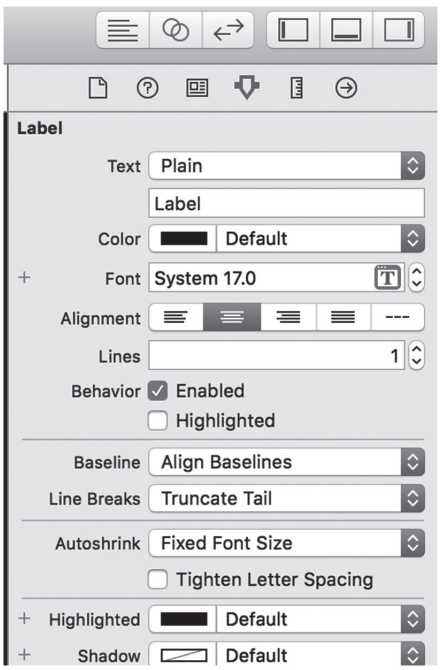

Рис. 3.13. Использование инспектора атрибутов метки для центрирования текста метки

Мы не хотим, чтобы на кнопке было что-нибудь написано, пока пользователь ее нажмет, поэтому дважды щелкните на метке (чтобы выбрать текст) и нажмите клавишу <Delete>. В результате текст, приписанный к кнопке в данный момент, будет удален. Нажмите клавишу <Return>, чтобы подтвердить исправления.

Даже если вы не видите метку, когда она не выбрана, не беспокойтесь — она на месте.

СОВЕТ. Если интерфейс содержит невидимые элементы, например пустые метки, и вы хотите их увидеть, выберите команду Canvas из меню Editor, а затем во всплывшем подменю установите флажок Show Bounds Rectangles. Если вы просто хотите выделить невидимый элемент, щелкните на пиктограмме в окне Document Outline.

Осталось только создать выход для метки. Эта процедура ничем не отличается от предыдущей. Откройте помощник редактора и файл `ViewController.swift`. Если возникнет необходимость переключать файлы, воспользуйтесь всплывающим меню на панели быстрого перехода, расположенным над помощником редактора.

Затем выберите метку в программе Interface Builder и, нажав клавишу <Control>, перетащите курсор от метки к заголовочному файлу. Установите его точно на требуемом методе действия. Увидев окна, показанные на рис. 3.14, отпустите кнопку мыши, и вы снова увидите всплывающее окно (см. рис. 3.9).

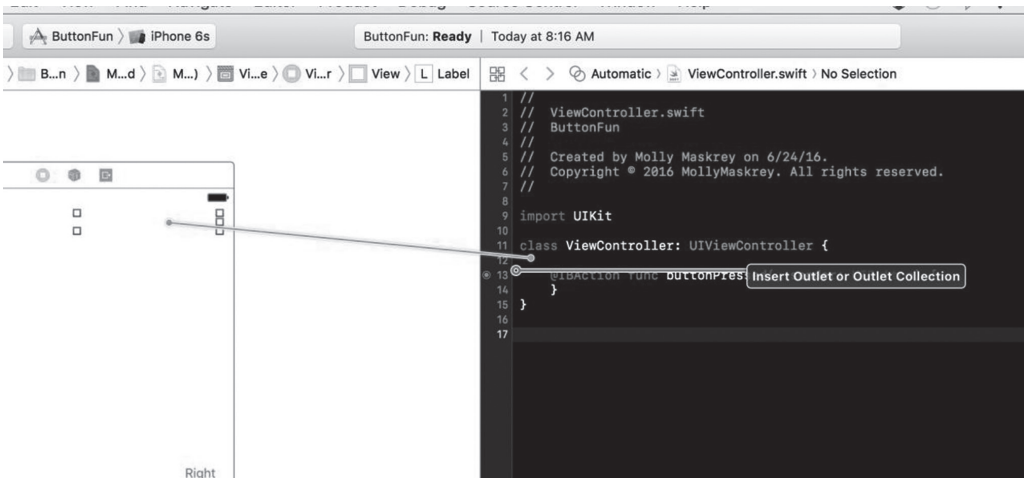

Рис. 3.14. Связывание выхода UILabel

Мы хотим создать выход, поэтому оставим тип `Connection` для объекта `Outlet`, заданный по умолчанию. Для того чтобы выбрать информативное имя для выхода, вспомним его назначение. Введите слово `statusLabel` в поле Name. Оставьте в поле Type значение UILable. В последнем поле Storage значение можно оставить по умолчанию.

Нажмите клавишу <Return>, чтобы подтвердить изменения, и программа Xcode вставит свойство выхода в ваш код. Заголовочный файл контроллера будет содержать код, представленный в листинге 3.6.

Листинг 3.6. Добавление выхода метки в класс `ViewController`

```
import UIKit

class ViewController: UIViewController {
   @IBOutlet weak var statusLabel: UILabel!
   @IBAction func buttonPressed(_ sender: UIButton) {
   }
}
```

Теперь у нас есть выход, и программа Xcode должна автоматически соединить с ним нашу метку. Это значит, что если мы изменим значение выхода `statusLabel` в коде, то это отобразится на метке в пользовательском интерфейсе. Если мы изменим свойство `text` для выхода `statusLabel`, то на экране изменится текст на метке.

АВТОМАТИЧЕСКИЙ ПОДСЧЕТ ССЫЛОК

Если вы знаете язык Objective-C или читали предыдущие издания настоящей книги, то могли заметить, что мы не использовали метод `dealloc`. Мы никогда не удаляли из памяти наши переменные экземпляров.

Компилятор LLVM, который компания Apple встроила в программу Xcode, настолько разумен, что освобождает объекты самостоятельно, используя новую функциональную возможность под названием Automatic Reference Counting (ARC).

Механизм ARC применяется только к объектам языка Swift и структурам, но не к объектам каркаса Core Foundation или объектам, размещенным в памяти с помощью функции `malloc()` или подобных ей функций. Есть еще несколько тонкостей и ловушек, но в целом управление памятью вручную осталось в прошлом.

Более полную информацию о механизме ARC можно найти по адресу `http://developer.apple.com/library/ios/#releasenotes/ObjectiveC/RNTransitioningToARC/`

Механизм ARC хорош, но не всемогущ. Необходимо хорошо понимать основные правила управления памятью в языке Objective-C, чтобы избежать неприятностей. Правила управления памятью в языке Objective-C можно найти в документе Memory Management Programming Guide, который компания Apple поместила на вебстранице `https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/MemoryMgmt/Articles/MemoryMgmt.html`.

Создание метода действия

Итак, мы разработали пользовательский интерфейс и связали между собой его выходы и действия. Осталось только применить эти действия и выходы для того, чтобы изменить текст на кнопке при ее нажатии. Щелкните мышью в окне навигатора проекта на файле `ViewController.swift`, чтобы открыть его. Найдите пустой метод `buttonPressed()`, созданный программой Xcode.

Для того чтобы наши кнопки отличались одна от другой, будем использовать параметр `sender`. Мы извлечем название нажатой кнопки из параметра `sender`, создадим строку на основе этого значения и присвоим его тексту метки. Изменим метод `buttonPressed()` так, как показано в листинге 3.7.

Листинг 3.7. Завершение метода действия

```
@IBAction func buttonPressed(sender: UIButton) {
   let title = sender.title(for: .selected)!
   let text = "\(title) button pressed"
   statusLabel.text = text
}
```

Это довольно просто. Первая инструкция этого фрагмента извлекает название кнопки из параметра `sender`. Поскольку кнопки могут иметь разные названия в зависимости от текущей ситуации, мы используем параметр `UIControlStateNormal`, чтобы указать, что нам нужно название кнопки в ее нормальном, не нажатом состоянии. Это типично для всех элементов управления (а кнопка — это один из элементов управления). Состояния элементов управления рассматриваются в главе 4.

СОВЕТ. Возможно, вы заметили, что при вызове метода `title(for:)` мы использовали аргумент `.selected`, а не `UIControlState.selected`. По правилам языка Swift аргумент должен быть одним из значений перечисления `UIControlState`, поэтому мы можем пропустить имя перечисления, чтобы сэкономить количество набираемого текста.

Следующая инструкция создает новую строку, добавляя к названию кнопки слова `button pressed`. Таким образом, если речь идет о левой кнопке, которая имеет название Left, то при ее нажатии эта строка программы создаст строку `Left button Pressed`. Эта новая строка присваивается свойству метки `text`. Вот так изменяется текст на метке при ее нажатии.

Тестирование приложения Button Fun

Выберите команду Product⇨Run. Если компилятор или редактор связей выдаст ошибки, вернитесь в окно редактирования и сравните свой код с текстом в главе. Если компиляция прошла без ошибок, то программа Xcode запустит симулятор устройства iPhone и выполнит приложение. Когда вы нажмете левую кнопку, то экран должен выглядеть так, как показано на рис. 3.15.

На первый взгляд, все в порядке, но если приглядеться, то обнаружится, что чего-то не достает. Для того чтобы увидеть, чего именно, измените текущую схему, как показано на рис. 3.16, на iPhone SE и снова запустите приложение.

Снимок экрана, показанный на рис. 3.17, свидетельствует о проблемах. Левая кнопка работает, как надо, но сдвинута вправо, а правая кнопка вообще исчезла.

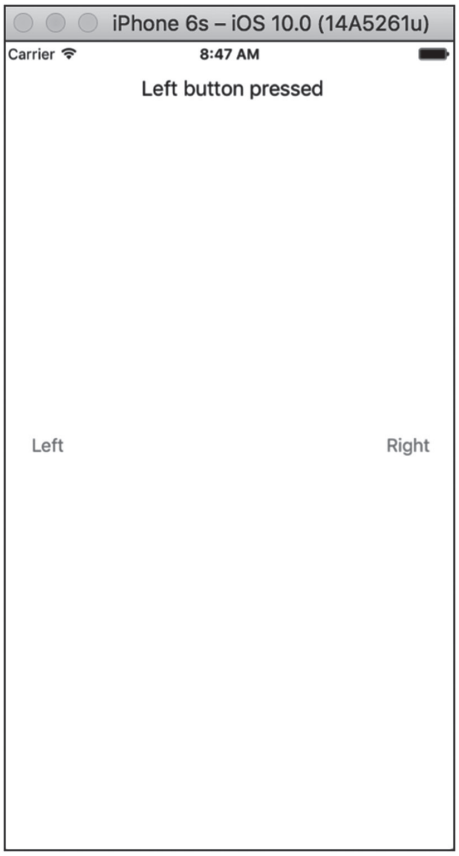

Рис. 3.15. Запуск приложение на iPhone 6s

Для того чтобы понять, почему это происходит, перейдите в среду Xcode и щелкните правой кнопкой в нижней части окна Interface Builder, чтобы выделить его и увидеть макет, а затем под окном выберите команду View As for iPhone SE, как показано на рис. 3.18. Поскольку мы настроили наш макет на устройство с более крупным экраном, при переходе на устройство с меньшим экраном некоторые элементы управления смещаются со своих позиций на новом дисплее.

Решение проблем с помощью механизма Auto Layout

Левая кнопка находится на правильном месте, а метка и другая кнопка — нет. В главе 2 мы исправили подобную проблему с помощью механизма Auto Layout. Идея механизма Auto Layout заключается в использовании ограничений, задающих место расположения элемента управления. В данном случае мы хотим добиться следующего.

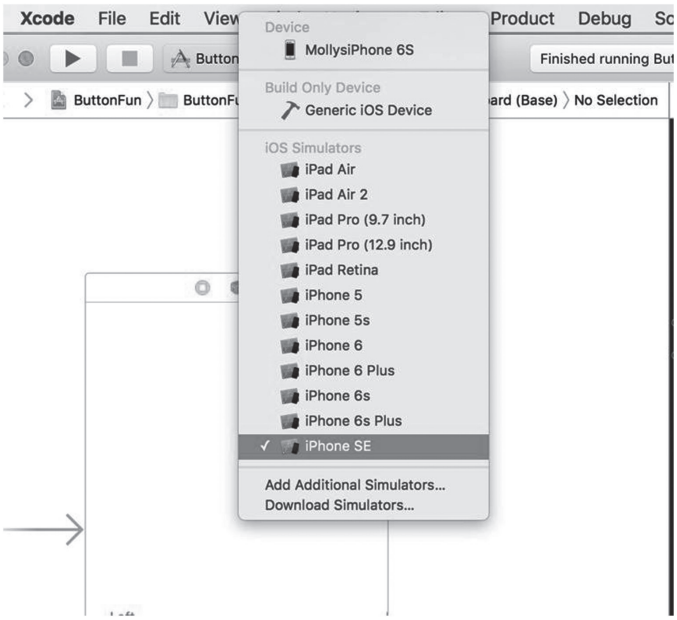

Рис. 3.16. Изменение схемы и целевого устройства на устройство с другими размерами и формой

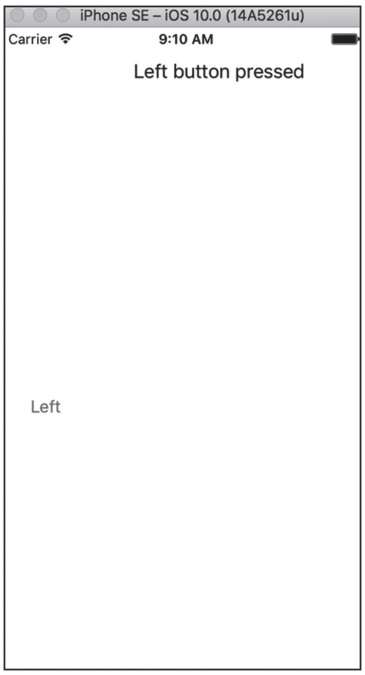

Рис. 3.17. На другом устройстве макет искажается

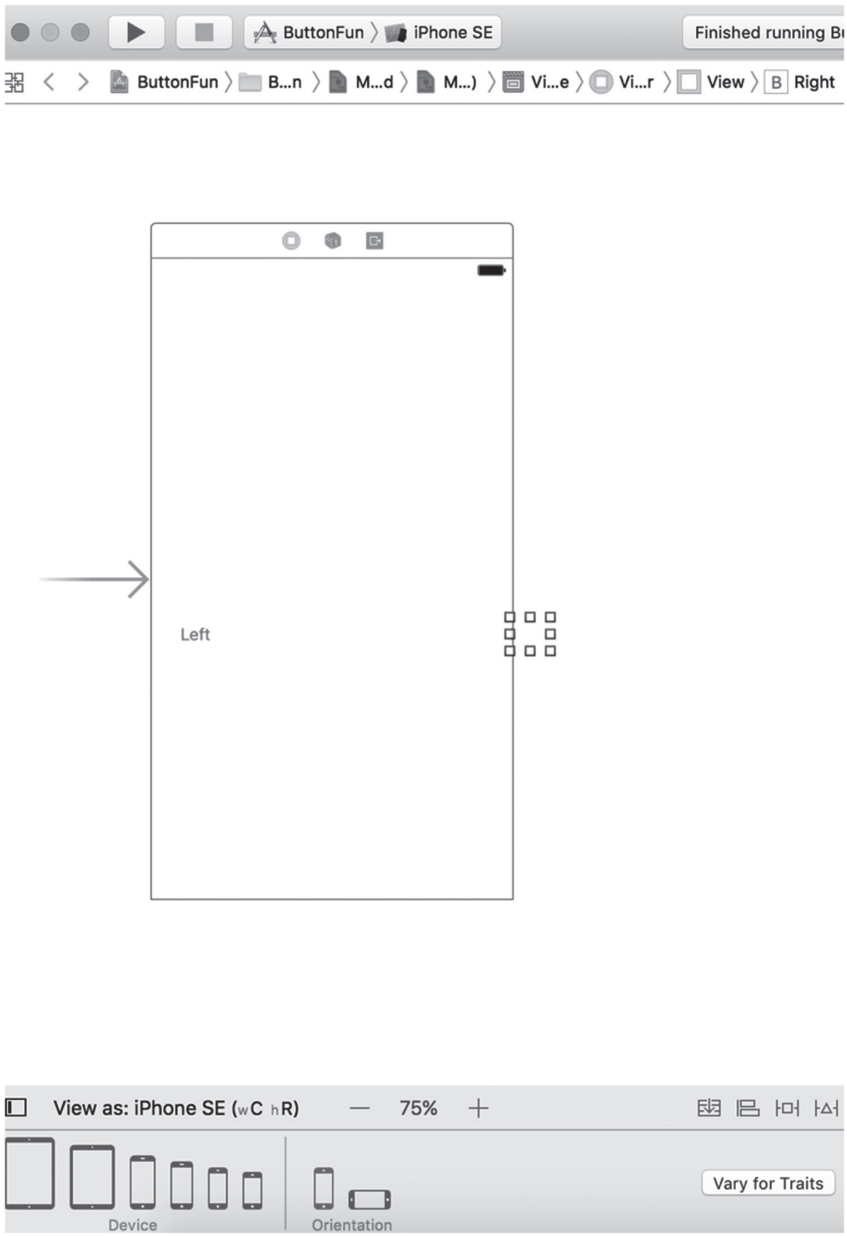

Рис. 3.18. На экране с меньшим экраном правая кнопка становится невидимой

- Кнопка Left должна быть отцентрована по вертикали и располагаться ближе к левому краю экрана.
- Кнопка Right должна быть отцентрована по вертикали и располагаться ближе к правому краю экрана.

- Метка должна быть отцентрована по горизонтали и немного отступать от верхнего края экрана.

Каждое из приведенных выше утверждений содержит два ограничения — одно касается ограничения по вертикали, а другие — по горизонтали. Если применить эти три ограничения ко всем трем нашим представлениям, то механизм Auto Layout автоматически разместит эти элементы в правильных местах любого экрана. Как же нам это сделать? Мы можем добавить ограничения механизма Auto Layout в представления, создав экземпляры класса `NSLayoutConstraint`. В некоторых ситуациях это единственный способ создания правильного макета, но в данном случае (как и во всех остальных примерах, приведенных в книге) правильный макет можно создать с помощью программы Interface Builder. Эта программа позволяет визуально добавлять ограничения с помощью перетаскивания и щелчков мышью. Перейдите в окно View As, расположенное под окном IB, и снова выберите устройство 6s в качестве целевого, чтобы увидеть все элементы управления. Затем задайте коэффициент масштабирования, например 75%. Механизм Auto Layout можно применять и для настройки макетов на другие устройства (см. рис. 3.19).

Начнем с позиционирования метки. Выберите пункт Main.storyboard в окне навигатора проекта и откройте окно Document Outline, чтобы увидеть иерархию представлений. Найдите пиктограмму с меткой View. Она символизирует контроллер главного представления по отношению к которому мы должны позиционировать другие представления. Щелкните на треугольнике раскрытия, чтобы открыть пиктограмму View, если она еще не открыта, и найдите две кнопки (с метками Left и Right) и метку. Проведите соединительную линию от метки к родительскому представлению, как показано на левой панели рис. 3.20.

Перетаскивая указатель мыши с одного представления на другое, вы сообщаете программе Interface Builder, что хотите применить к ним ограничения механизма Auto Layout. Отпустите кнопку мыши — и на экране появится серое всплывающее окно с множеством команд (рис. 3.20). Каждая из этих команд представляет собой отдельное ограничение. Выбрав любую из этих команд, вы применяете соответствующее ограничение. Однако, как нам известно, нам необходимо применить к метке два ограничения, причем оба они есть в списке команд всплывающего меню. Для того чтобы применить сразу несколько ограничений, необходимо нажать и удерживать клавишу Shift, выбирая соответствующие команды. Итак, нажмите клавишу Shift и выберите команды Center Horizontally in Container и Vertical Spacing to Top Layout Guide. Для того чтобы эти ограничения были действительно применены, щелкните мышью в любом месте за пределами всплывающего меню или нажмите клавишу <Return>. После этого созданные вами ограничения появятся в разделе Constraints в окне Document Outline, а также будут представлены визуально в раскадровке, как показано на рис. 3.21.

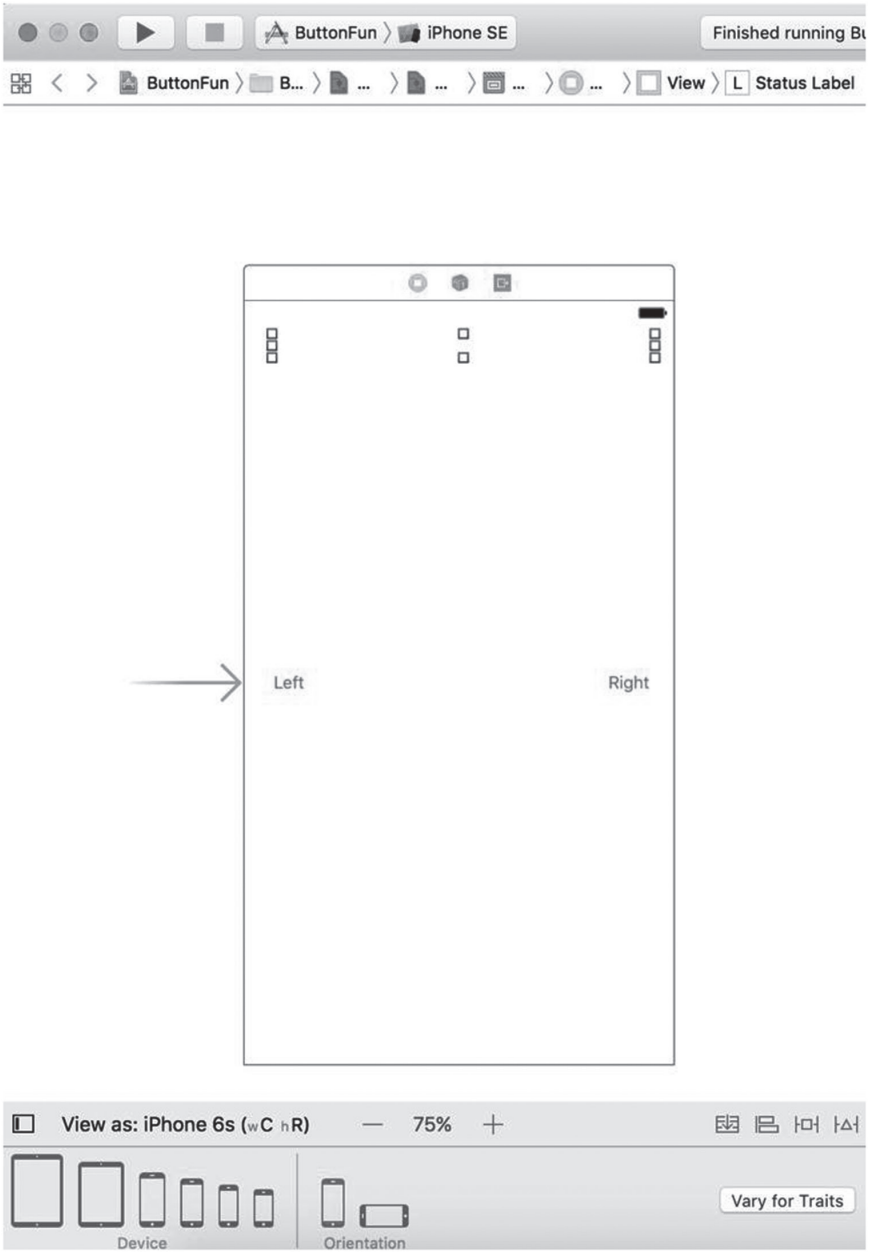

Рис. 3.19. Для настройки макета на другие устройства с помощью механизма Auto Layout мы используем то устройство, для которого мы изначально создавали свое приложение

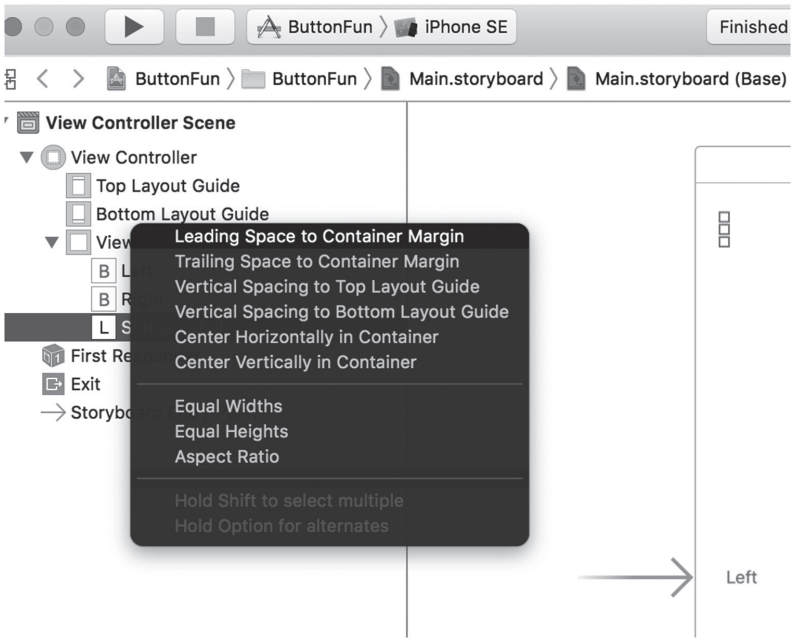

Рис. 3.20. Позиционирование метки с помощью ограничений Auto Layout

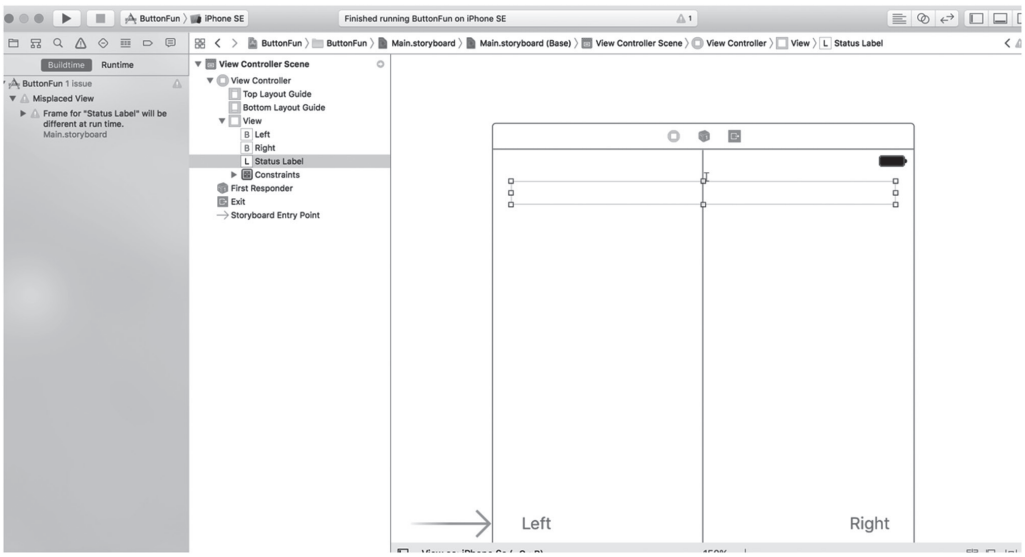

Рис. 3.21. Два ограничения Auto Layout, применяемые к метке

ПОДСКАЗКА. Если, создавая ограничение, вы сделали ошибку, удалите его, щелкнув на его представлении в окне Document Outline, или удалите его из раскадровки, нажав клавишу <Delete>.

Вероятно, вы заметили, что кнопка имеет оранжевый контур. Этим цветом программа Interface Builder отмечает возникновение проблемы в механизме Auto Layout. Существует три вида типичных проблем, о которых сообщает программа Interface, рисуя оранжевый контур.

- Вы задали недостаточное количество ограничений для того, чтобы точно указать позицию или размер представления.
- Вы задали неоднозначные ограничения, т.е. они задают размер или позицию не единственным образом.
- Ограничения являются правильными, но позиция и/или размер представления во время выполнения программы не совпадает с соответствующей позицией и/или размером в программе in Interface Builder.

Получить более подробную информацию о проблеме можно, щелкнув на оранжевом треугольнике в окне Activity View в навигаторе проблем (см. левую часть рис. 3.21). В результате вы увидите строку “Frame for ‘Label’ will be different at run time” (Рамка объекта ’Label’ во время выполнения приложения будет другой), сообщающую о проблеме третьего вида. Это сообщение можно стереть, сделав так, чтобы программа Interface Builder переместила метку на правильную позицию и задала ее правильные размеры. Для этого обратите внимание на редактор раскадровок, расположенный в правом нижнем углу. Вы видите четыре кнопки, показанные на рис. 3.22.

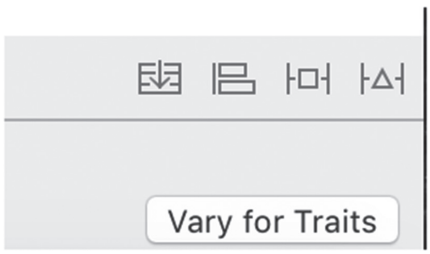

Рис. 3.22. Кнопки механизма Auto Layout в правом нижнем углу редактора раскадровок

Вы можете узнать, что делает каждая из этих кнопок, задержав указатель мыши над ними. Левая кнопка связана с элементом управления UIStackView, которую мы будем рассматривать в главе 10. Переходя слева направо, перечислим их функции.

1. Кнопка Align позволяет выровнять выбранное представление относительно другого представления. Щелкнув на этой кнопке, вы увидите всплывающее

меню, содержащее разные варианты выравнивания. Одна из этих кнопок — Horizontal Center in Container — соответствует ограничению, которое уже применялось к метке в окне Document Outline. Как правило, одну и ту же функцию механизма Auto Layout в программе Interface Builder можно выполнить несколькими способами. По мере чтения книги вы узнаете альтернативные способы выполнения задач, связанных с использованием механизма Auto Layout.

2. Всплывающее меню для кнопки Pin содержит команды, позволяющие задавать позицию представления относительно других представлений и применять ограничения размеров. Например, можно задать ограничение, требующее, чтобы высота одного представления совпадала с высотой другого.
3. Кнопка Resolve Auto Layout Issues позволяет решать проблемы, связанные с макетом. Меню, соответствующее этой кнопке, содержит команды, позволяющие удалять все ограничения, установленные для представления (или всю раскадровку), выяснять, какие ограничения пропущены, и добавлять их, а также уточнять рамки одного или нескольких представлений, которые должны применяться на этапе выполнения приложения.

Изменить рамку метки можно, выбрав ее в окне Document Outline или в раскадровке и щелкнув на кнопке Resolve Auto Layout Issues. Всплывающее меню для этой кнопки содержит две идентичные группы операций (рис. 3.23).

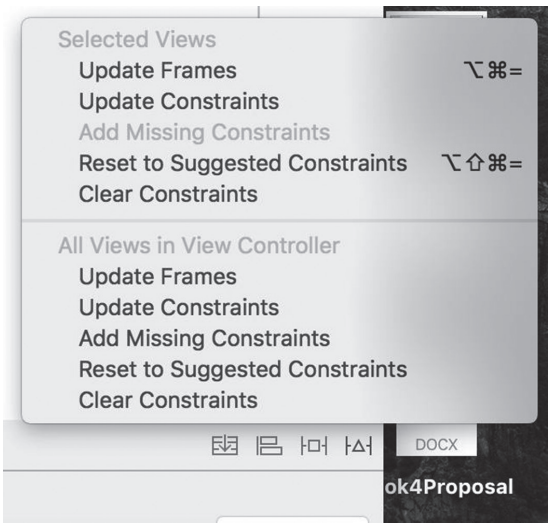

Рис. 3.23. Всплывающее меню для кнопки Resolve Auto Layout Issues

ПОДСКАЗКА. Если все команды всплывающего меню недоступны, щелкните на метке в окне Document Outline, чтобы обеспечить возможность их выбора.

Команды из верхней части меню относятся только к текущему представлению, а команды из нижней части меню — ко всем представлениям, связанным с контроллером представлений. В данном случае нам просто необходимо исправить рамку метки, поэтому мы выберем команду Update Frames в верхней части меню. После этого выберем оранжевый контур и треугольник предупреждения в окне Activity View, потому что теперь во время выполнения программы метка будет находиться на правильной позиции и иметь правильный размер. Фактически ширина метки будет уменьшена до нуля и будет представлена в раскадровке как маленький пустой квадратик (рис. 3.24).

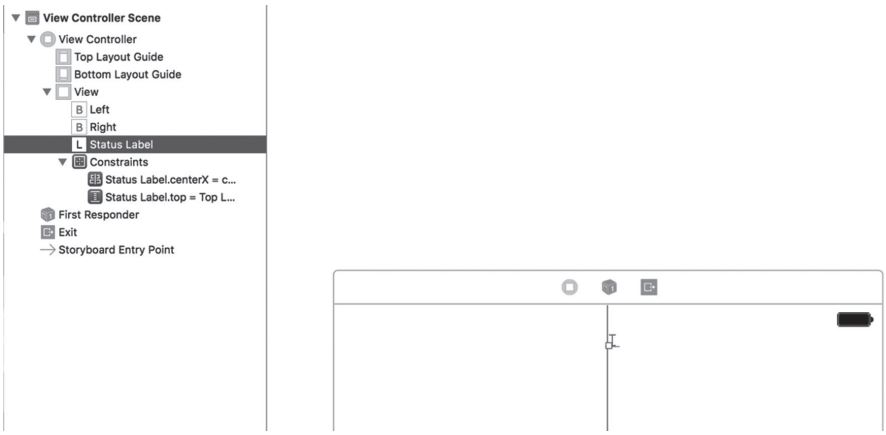

Рис. 3.24. После исправления контура ширина кнопки уменьшилась до нуля

Можно ли это считать правильным? Оказывается, можно. Многие из представлений, поставляемых в библиотеке UIKit, включая класс `UILabel`, позволяют механизму Auto Layout вычислять их размер по их реальному содержимому. Это возможно благодаря вычислению естественного (natural), или действительного (intrinsic), размера их содержимого. С точки зрения действительного размера ширина и высота кнопки вполне достаточны, чтобы на ней поместился текст ее названия. Пока у кнопки нет названия, высота и ширина кнопки действительно равны нулю. Когда мы запустим приложение и щелкнем на одной из кнопок, ее действительный размер будет изменен и мы увидим весь текст.

Исправив недостатки метки, перейдем к уточнению позиций двух кнопок. Выберите кнопку Left на раскадровке и щелкните на кнопке Align в правом нижнем углу окна редактора раскадровки (последняя слева кнопка на рис. 3.22). Мы хотим, чтобы кнопка была отцентрована по вертикали, поэтому выберем команду Vertical Center in Container во всплывающем меню и щелкнем на кнопке Add 1 Constraint (рис. 3.25).

Это же ограничение необходимо применить к кнопке Right, поэтому выберем ее и повторим описанный процесс. В этом случае программа Interface Builder выявит несколько новых проблем, которые будут выделены оранжевым цветом

в раскадровке и отмечены треугольником предупреждения в окне Activity View. Щелкните на этом треугольнике, чтобы узнать о причинах ошибок, обнаруженных навигатором проблем (рис. 3.26).

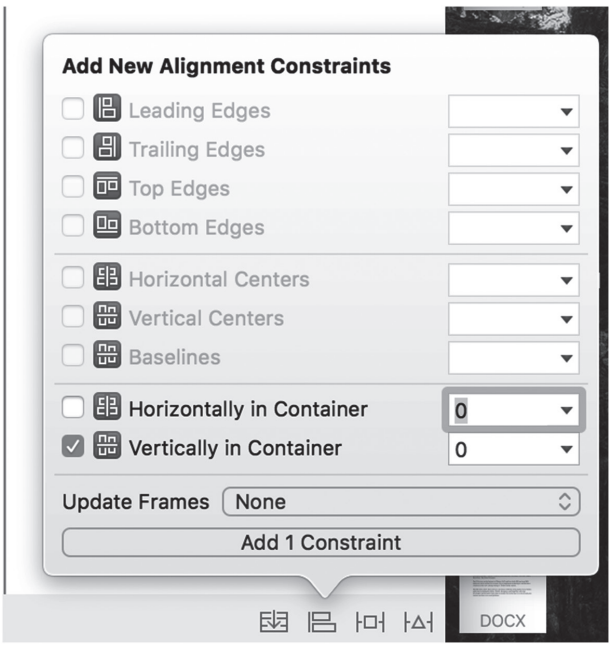

Рис. 3.25. Центрование представления по вертикали с помощью всплывающего меню Align

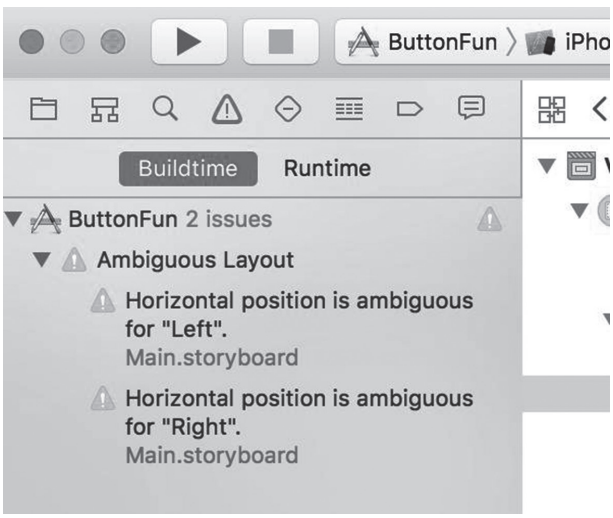

Рис. 3.26. Предупреждение программы Interface Builder об отсутствии ограничений

Программа Interface Builder предупреждает о том, что горизонтальные позиции обеих кнопок заданы неоднозначно. Фактически у нас нет ограничений, управляющих горизонтальными позициями кнопок, поэтому предупреждение не должно нас удивлять.

ЗАМЕЧАНИЕ. Задавая ограничения Auto Layout, вы будете часто получать подобные предупреждения. Они необходимы для того, чтобы вы задали полный набор ограничений. По завершении процесса разметки вы не должны получить ни одного предупреждения. Большинство примеров в этой книге содержат инструкции для задания ограничений разметки. Добавляя эти ограничения, вы, конечно, должны учитывать предупреждения, но беспокоиться стоит, только если они появляются после завершения всего процесса разметки. Это будет значить, что вы пропустили какой-то шаг, выполнили его неправильно или в книгу вкралась ошибка! В последнем случае, пожалуйста, сообщите об ошибке по адресу `www.apress.com`.

Мы хотим, чтобы кнопка Left находилась на фиксированном расстоянии от левого края родительского представления, а кнопка Right — на том же расстоянии от его правого края. Эти ограничения можно задать с помощью всплывающего меню, связанного с кнопкой Pin (следующей за кнопкой Align на рис. 3.22). Выберите кнопку Left и щелкните на кнопке Pin, чтобы открыть ее всплывающее меню. В верхней части меню вы найдете четыре поля редактирования, связанных с маленьким квадратиком с помощью оранжевых пунктирных линий, показанных на левой части рис. 3.27. Маленький квадратик представляет кнопку, для которой задаются ограничения. Четыре поля редактирования позволяют задать расстояние он кнопки от ближайших соседей над и под ней, а также слева и справа от нее. Пунктирная линия означает, что соответствующего ограничения пока нет. Мы хотим, чтобы кнопка Left находилась на фиксированном расстоянии от левого края своего родительского представления, поэтому щелкнем на пунктирной оранжевой линии, идущей влево от квадрата. В этом случае она станет сплошной оранжевой линией, означающей, что соответствующее ограничение уже установлено. Затем введем число `32` в левое поле редактирования, чтобы задать расстояние от кнопки Left до его родительского представления. Всплывающее меню должно стать таким, как показано на правой части рис. 3.22. Нажмите кнопку Add 1 Constraint, чтобы применить это ограничение к данной кнопке.

Для того чтобы исправить позицию кнопки Right, выберите ее, нажмите кнопку Pin, щелкните на пунктирной оранжевой линии, идущей вправо от квадрата (поскольку мы хотим зафиксировать расстояние от кнопки до правого края ее родительского представления), введите число `32` в поле редактирования и нажмите кнопку Add 1 Constraint.

Теперь мы применили все необходимые ограничения, но предупреждения в окне Activity View не исчезли. Анализ показывает, что на этапе выполнения приложения кнопки будут располагаться неправильно. Для того чтобы исправить эту проблему, необходимо нажать кнопку Resolve Auto Layout Issues. В результате

откроется всплывающее меню, в котором надо выбрать команду Update Frames в нижнем разделе. Это ограничение будет касаться рамок всех представлений в настраиваемом контроллере представлений.

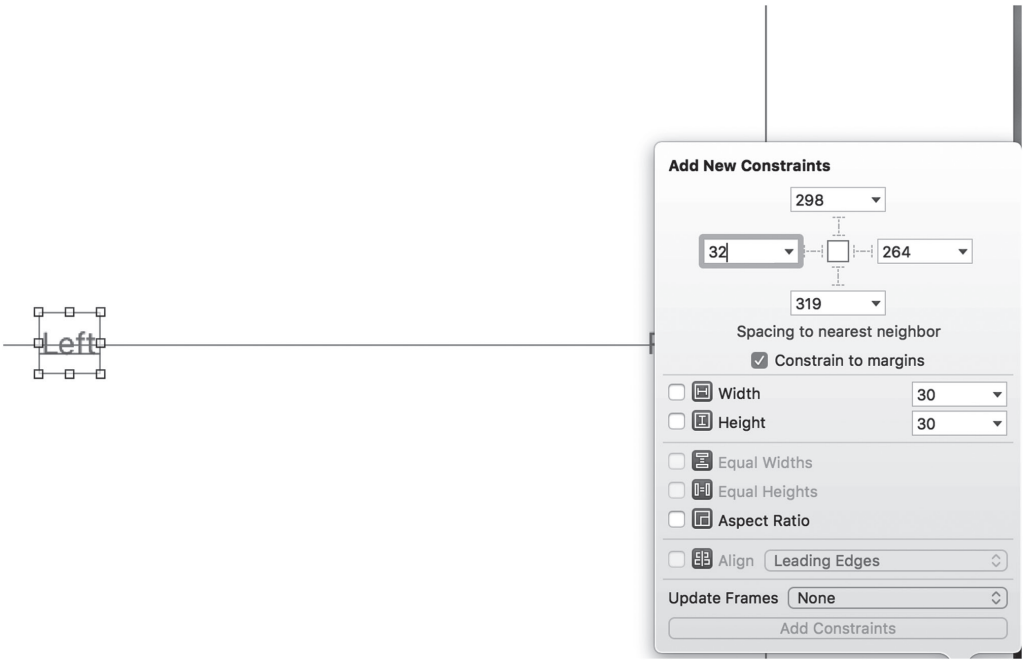

Рис. 3.27. Выравнивание представления по горизонтали с помощью всплывающего меню Pin

ПОДСКАЗКА. Иногда во всплывающем меню не все команды являются доступными. В этом случае выберите пиктограмму View Controller в окне Document Outline и попытайтесь снова.

Теперь предупреждения должны исчезнуть и разметка будет, наконец, завершена. Запустите свое приложение на симуляторе устройства iPhone — и увидите результат, очень похожий на рис. 3.1, помещенный в начале главы. Коснувшись правой кнопки, вы должны увидеть заголовок кнопки Right button pressed. Коснувшись левой кнопки, вы должны увидеть заголовок кнопки Left button pressed. Запустите приложение на симуляторе iPad, и обнаружите, что макет все еще работает, хотя кнопки расположены далеко одна от другой, потому что экран iPad шире, чем экран iPhone.

ПОДСКАЗКА. При запуске приложения на имитируемых устройствах с большими экранами иногда невозможно увидеть весь экран сразу. Эту проблему можно исправить, выбрав команду Window⇨Scale в меню программы iOS Simulator и выбрав коэффициент масштабирования для своего экрана.

Взглянув на рис. 3.1, вы увидите, что одну вещь мы пропустили. Заглавие выбранной кнопки должно выводиться на экран полужирным шрифтом, а пока мы видим обычный текст. Немного позже мы исправим это с помощью класса `NSAttributedString`, а сначала ознакомимся с другой полезной функцией среды Xcode — предварительным просмотром макета.

Предварительный просмотр макета

Вернитесь в программу Xcode и выберите узел Main.storyboard, а затем откройте окно помощника редактора, если оно закрыто (как это сделать, показано на рис. 3.6). В левой половине панели быстрых переходов, расположенной в верхней части окна помощника редактора, вы увидите, что в данный момент выбран атрибут Automatic (если вы не изменили его на Manual, чтобы выбрать файл в окне помощника редактора). Щелкните на сегменте панели быстрых переходов, чтобы открыть всплывающее меню, и увидите несколько команд, последняя из которых называется Preview. Если установить на нее указатель мыши, появится меню, содержащее имя раскадровки приложения. Щелкните на нем, чтобы открыть раскадровку в окне Preview Editor.

Когда откроется окно Preview Editor, вы увидите внешний вид приложения на устройстве iPhone в книжной ориентации. Это всего лишь предварительный просмотр, поэтому никаких реакций на касание кнопок нет, а значит, вы не увидите метку. Если вы переместите мышь на область, расположенную ниже области предварительного просмотра, где написано `iPhone 6s`, появится элемент управления, позволяющий поворачивать телефон и переводить экран в альбомную ориентацию. Этот элемент управления изображен в левой части рис. 3.28. Щелкнув на нем, вы выполните вращение телефона по часовой стрелке.

Благодаря механизму Auto Layout при вращении телефона кнопки перемещаются так, чтобы их центровка и расстояния были такими, как в книжной ориентации. Если метка остается видимой, то ее позиция также будет правильной.

Кроме того, помощник предварительного просмотра можно использовать для того, чтобы увидеть, что произойдет, если запустить приложение на другом устройстве. В левом нижнем углу окна помощника предварительного просмотра (и на рис. 3.28) вы увидите кнопку +. Щелкните на ней, чтобы открыть список устройств, а затем выберите пункт iPad, чтобы иметь возможность предварительного просмотра экрана iPad в окне помощника предварительного просмотра. Экран iPad занимает много места, поэтому, возможно, вам придется закрыть окна Document Outline и Utility View, чтобы можно было видеть и экран iPhone, и экран iPad. Если вы все еще не видите полный экран устройства iPad, то можете масштабировать окно Preview Assistant, используя разные способы. Проще всего дважды щелкнуть на панели Preview Assistant — и оно станет значительно меньше. Если вы хотите задать конкретный коэффициент масштабирования, то можете выполнить жест щипка на мультисенсорной панели (к сожалению, мышь Magic Mouse эту возможность не поддерживает). Окна предварительного просмотра экранов iPhone и iPad, уменьшенные так, чтобы их было видно

полностью, показаны на рис. 3.29. Обратите внимание на то, что механизм Auto расположил кнопки правильно. Поверните окно предварительного просмотра экрана iPad, чтобы убедиться в том, что приложение правильно работает в альбомном режиме.

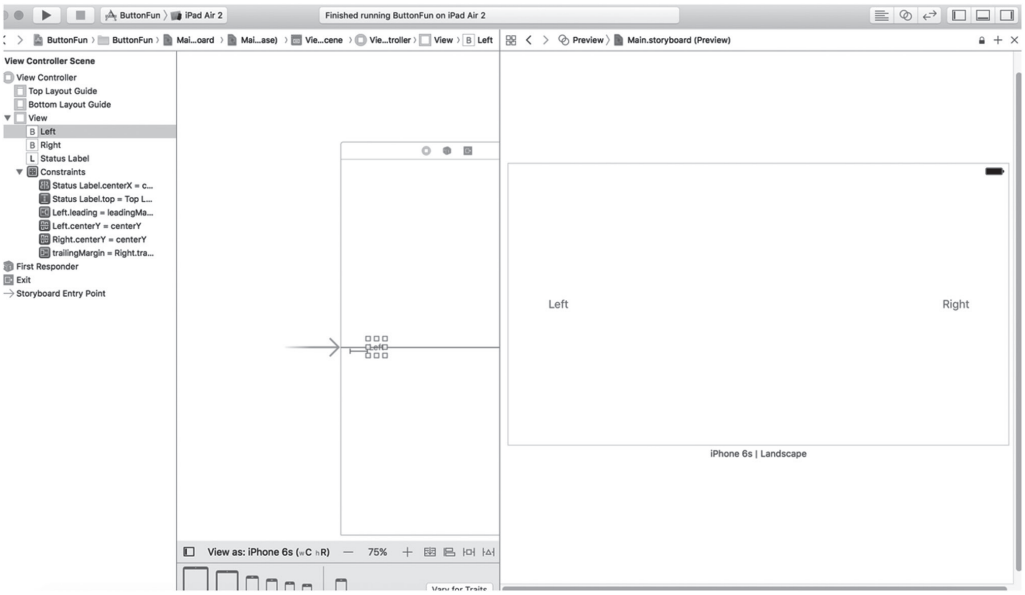

Рис. 3.28. Предварительный просмотр разметки на экране устройства iPhone альбомной ориентации

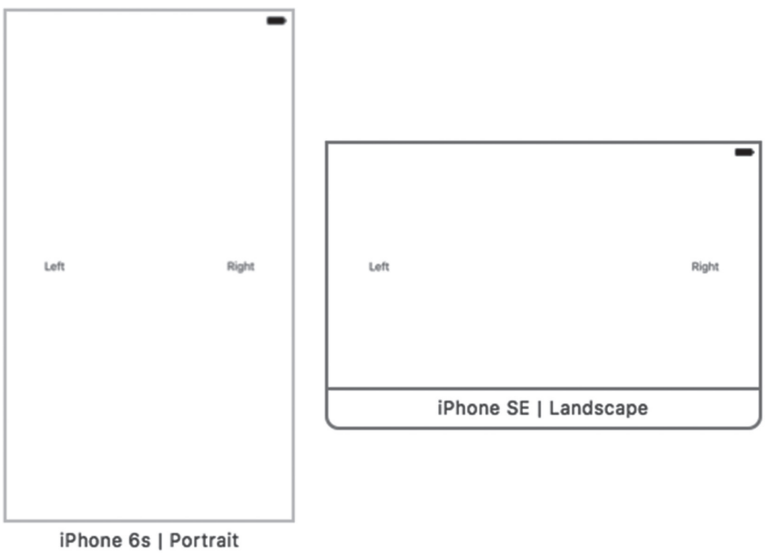

Рис. 3.29. Одновременный предварительный просмотр экранов iPhone и iPad

ЗАМЕЧАНИЕ. Когда мы работали над книгой, панель Preview Assistant некорректно отображала макеты для iPad с экранами 9,7 и 12,9 дюйма. Возможно, это связано с ошибками бета-версии Xcode 8. Однако выполнение соответствующих приложений на симуляторе показывает, что они работают корректно.

Изменение стиля текста

Класс `NSAttributedString` позволяет добавлять информацию о формате, например о шрифтах и выравнивании абзацев. Эти метаданные можно применять ко всей строке, причем разные атрибуты можно применять к разным частям интерфейса. Для того чтобы понять, как работает класс `NSAttributedString`, достаточно вспомнить, как форматируется текст в текстовом редакторе. Большинство элементов управления в библиотеке UIKit позволяют использовать строки с атрибутами. В случае класса `UILabel`, который мы используем в своем приложении, можно просто создать строку с атрибутами и передать ее метке с помощью ее свойства `attributedText`.

Итак, выберем файл `ViewController.swift` и обновим метод `buttonPressed()` так, как показано в листинге, удалив перечеркнутую строку и добавив строки, приведенные в листинге 3.8.

Листинг 3.8. Обновление метода `buttonPressed()` для полужирного шрифта

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var statusLabel: UILabel!

    @IBAction func buttonPressed(_ sender: UIButton) {
        let title = sender.title(for: .selected)!
        let text = "\(title) button pressed"
        let styledText = NSMutableAttributedString(string: text)
        let attributes = [
            NSFontAttributeName:
                UIFont.boldSystemFont(ofSize: statusLabel.font.pointSize)
        ]
        let nameRange = (text as NSString).range(of: title)
        styledText.setAttributes(attributes, range: nameRange)

        statusLabel.attributedText = styledText
    }
}
```

Сначала новый код создает строку с атрибутами, в частности объект класса `NSMutableAttributedString`, на основе строки, которую вы хотите вывести на экран. Нам нужна строка с атрибутами, допускающая изменения, потому что мы собираемся изменять ее атрибуты.

Далее мы создаем словарь для хранения атрибутов, которые будут применяться к строке. На самом деле у нас пока только один атрибут, поэтому словарь содержит единственную пару "ключ–значение". Ключ `NSFontAttributeName` позволяет задать шрифт для части строки с атрибутами. Значение, которое мы передаем, иногда называют "полужирный системный шрифт". Оно означает, что размер шрифта строки с атрибутами должен совпадать с размером шрифта, который в данный момент используется для метки. Такое задание шрифта является более гибким, чем указание шрифта по названию, поскольку система сама знает, как правильно использовать полужирный шрифт.

Теперь мы попросим строку `plainText` сообщить нам диапазон (состоящий из начального индекса и длины) подстроки, содержащей название метки. Применим эти атрибуты к строке с атрибутами и передадим ее метке. Рассмотрим строку, задающую координаты строки заголовка.

```
let nameRange = (text as NSString).range(of: title)
```

Обратите внимание на то, что тип переменной `plainText` приводится из типа `String` языка Swift в тип `NSString` каркаса Core Foundation. Это необходимо, потому что оба класса, `String` и `NSString`, имеют метод `range(of: String)`. Метод класса `NSString` задает диапазон в виде объекта `NSRange`, потому что именно его ожидает метод `setAttributes()` в следующей строке программы.

Теперь можно щелкать на кнопке Run; вы увидите, что приложение показывает название нажатой кнопки с помощью полужирного шрифта, как на рис. 3.1.

Использование делегата приложения

Теперь, когда наше приложение работает, прежде чем переходить к новой теме, уделим несколько минут изучению исходного файла, который мы еще не просматривали: `AppDelegate.swift`. Этот файл является реализацией делегата приложения (application delegate).

Делегаты широко используются в каркасе Cocoa Touch. Они представляют собой классы, решающие определенные задачи от имени другого объекта. Делегат приложения позволяет выполнять определенные действия в заранее установленное время от имени класса `UIApplication`. Каждое приложение для системы iOS имеет один и только один экземпляр класса `UIApplication`, обеспечивающий выполнение приложения и реализующий его функциональные возможности, такие как направление входных данных соответствующему классу контроллера. Класс `UIApplication` является стандартной частью библиотеки UIKit и выполняет свою работу практически незаметно, так что в большинстве случаев о нем не приходится беспокоиться.

В определенные точно заданные моменты времени в ходе выполнения приложения класс `UIApplication` вызывает установленные методы делегата, при условии, что делегат, реализующий этот метод, действительно существует. Например,

если у вас есть код, который должен сработать непосредственно перед завершением программы, можете реализовать метод `applicationWillTerminate()` в делегате своего приложения и поместить этот код в него. Данный тип делегирования позволяет вашему приложению реализовать обычное поведение без создания подкласса класса `UIApplication` и даже без информации о том, как он работает.

Щелкните на файле `AppDelegate.swift` в окне навигатора проекта, и увидите содержимое заголовочного файла делегата своего приложения (листинг 3.9).

Листинг 3.9. Первоначальный код делегата приложения

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

   var window: UIWindow?
```

Выделенная полужирным шрифтом часть строки означает, что класс поддерживает протокол `UIApplicationDelegate`. Нажмите и удерживайте клавишу <Option>. Ваш курсор имеет вид ножниц. Переместите курсор на слово `UIApplicationDelegate`. Он имеет вид знака вопроса, а слово `UIApplicationDelegate` будет выделено как ссылка браузера (рис. 3.30).

Рис. 3.30. После того как вы нажали клавишу <Option> в среде Xcode и указали на символ в нашем коде, этот символ стал выделенным, а курсор принял вид знака вопроса

Продолжая удерживать нажатой клавишу <Option>, щелкните на этой ссылке. Откроется маленькое всплывающее окно с кратким описанием протокола `UIApplicationDelegate` (рис. 3.31).

Проходя по всплывающему меню сверху вниз, вы найдете две ссылки (см. рис. 3.32).

Обратите внимание на две ссылки, расположенные в нижней части всплывающего окна документации. Щелкните на ссылке More, чтобы увидеть полную документацию для этого символа, или на ссылке Declared, чтобы увидеть определение символа в заголовочном файле. Аналогичный трюк работает и для имен классов, протоколов и категорий, а также для методов, отображаемых на панели редактирования. Просто дважды щелкните на слове, и программа Xcode найдет для вас это слово в браузере документации.

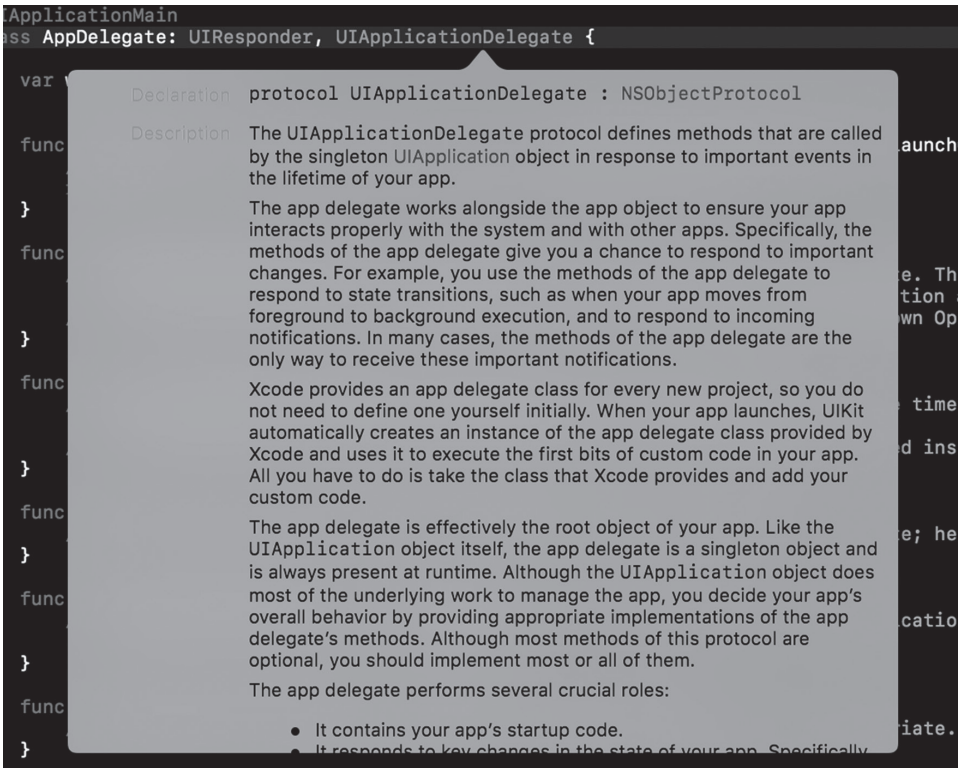

Рис. 3.31. После того как вы нажали клавишу <Option> и щелкнули на ссылке UIApplicationDelegate в исходном коде, программа Xcode открыла панель Quick Help с описанием требуемого протокола

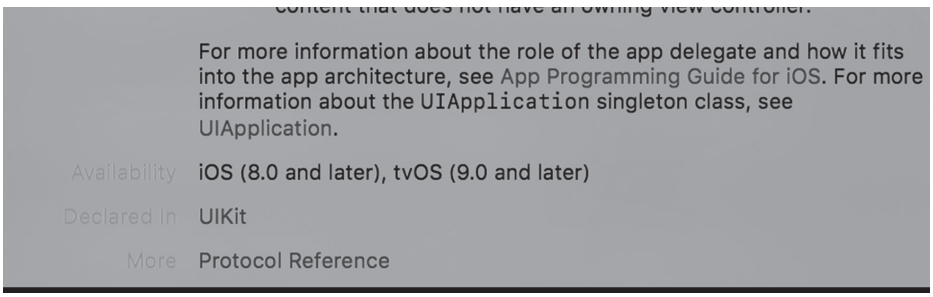

Рис. 3.32. Ссылки на дополнительную информацию о выделенном элементе

Умение быстро находить нужную информацию о протоколе в документации, безусловно, важно, но еще важнее иметь возможность видеть определение протокола. Именно здесь вы можете узнать, какие методы делегата приложения можно реализовать и когда эти методы будут вызваны. Вероятно, стоит потратить время на изучение описания этих методов.

Вернитесь в окно навигатора проекта и щелкните на файле `AppDelegate.swift`, чтобы увидеть делегат приложения. Содержимое файла должно выглядеть так, как показано в листинге 3.10.

Листинг 3.10. Файл AppDelegate.swift

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

   var window: UIWindow?

   func application(_ application: UIApplication,
         didFinishLaunchingWithOptions
         launchOptions: [NSObject: AnyObject]?) -> Bool {
         // Точка замещения для настройки после запуска приложения.
         return true
   }

   func applicationWillResignActive(_ application: UIApplication) {
        // Вызывается, если приложение должно перейти из активного
        состояния
        в неактивное. Эта необходимость возникает при выполнении
        прерываний определенного типа (например, при входящем звонке или SMS)
        или когда пользователь выходит из приложения и выполняет какие-то
        действия в фоновом режиме.
        // Этот  метод используется для остановки выполняемых задач,
        отключения таймеров и замедления прорисовки кадров OpenGL ES, а также
        в играх для организации паузы.
   }

   func applicationDidEnterBackground(_ application: UIApplication) {
        // Этот метод используется для освобождения общих ресурсов,
       сохранения пользовательских данных, обнуления таймеров и сохранения
       информации о состоянии приложения, достаточно для его возобновления
       // Если ваше приложение поддерживает работу в фоновом режиме,
       этот метод вызывается при выходе пользователя вместо метода
       applicationWillTerminate:.
   }

   func applicationWillEnterForeground(application: UIApplication) {
      // Этот метод вызывается как часть перехода из фонового
      состояния в активное; здесь можно отменить изменения,
      сделанные в фоновом режиме.
   }

   func applicationDidBecomeActive(application: UIApplication) {
      // Возобновляет выполнение остановленных задач (или запускает
      еще не стартовавшие), оставляя приложение неактивным. Если приложение
      работало в фоновом режиме, пользовательский интерфейс может быть
      прорисован заново.
   }
```

```
    func applicationWillTerminate(application: UIApplication) {
        // Вызывается во время прекращения работы приложения.
        Сохраняет данные, если есть возможность.
        См. также applicationDidEnterBackground:.
    }
}
```

В начале файла можно увидеть, что делегат нашего приложения реализовал один из методов протокола `application(_: didFinishLaunchingWithOptions:)`, который, как легко догадаться, выполняется, как только приложение завершает этап настройки и готово к взаимодействию с пользователем. Этот метод часто используется для создания любых объектов, которые должны существовать на всем протяжении выполнения приложения.

В остальных частях книги, особенно в главе 15, мы узнаем, насколько важную роль играют делегаты в жизни приложений. Мы просто хотели кратко описать делегаты и показать, как тесно они связаны между собой.

Резюме

Простое приложение, рассмотренное в настоящей главе, позволило вам ознакомиться с концепцией MVC, создать и связать между собой выходы и действия, реализовать контроллеры представлений и использовать делегаты приложений. Вы научились инициировать действия при нажатии кнопки и узнали, как изменить метку кнопки во время выполнения программы. Несмотря на простоту созданного приложения, основные концепции, рассмотренные нами, совпадают с концепциями, лежащими в основе всех других элементов управления в системе iOS, а не только кнопок. Способ использования кнопок и меток, продемонстрированный в главе, прекрасно работает и со всеми другими элементами управления в системе iOS.

Чрезвычайно важно, чтобы вы поняли, что и почему мы делали в этой главе. Если это не так, то вернитесь к началу и повторяйте все действия, пока не поймете. Если вы не разобрались во всех деталях, то еще больше запутаетесь при создании более сложных интерфейсов в последующих главах книги.

В следующей главе мы изучим стандартные элементы управления iOS. Вы также узнаете, как выдавать предупреждения и уведомления пользователям и предоставлять им выбор с помощью листов действий.