Глава 10

Механизм Storyboards

В предыдущих главах мы детально рассмотрим nib-файлы, класс `UITableView` и механизм навигации между представлениями с помощью класса `UINavigationViewController`. В совокупности эти конструкции образуют надежный и гибкий набор инструментов для создания мобильных приложений, о чем свидетельствуют сотни тысяч приложений для устройств iPhone и iPad, разработанных за последние несколько лет.

Однако нет ничего идеального. По мере того как этими инструментами овладевало все больше и больше людей, их недостатки стали ощущаться все сильнее. О некоторых из них вы, возможно, уже знаете.

- Шаблон delegate/dataSource для определения содержимого объекта класса `UITableView` очень хорошо подходит для создания динамических таблиц, но если вы заранее точно знаете, что именно будет содержать таблица, то этот шаблон становится слишком громоздким. Что было бы, если бы могли определить содержимое таблицы в более декларативном стиле, пропустив все вызовы и ответы, представляющие собой ограничения операционной системы?
- Использование nib-файлов для хранения сублимированных графов объекта выше всяческих похвал, пока дело касается только этого. Но если ваше приложение имеет несколько контроллеров представлений, как почти все приложения, то переключение между ними всегда требует определенного объема ручной работы в виде избыточного шаблонного кодирования. Нельзя ли обойтись без этого?
- В сложном приложении с многочисленными контроллерами представлений трудно охватить взглядом всю картину. Связи и переходы между контроллерами скрываются в каждом классе контроллера. Это не только затрудняет чтение исходного кода приложения, требуя изучения взаимосвязей между делегатами и методами действий всех контроллеров, но и делает код приложения очень ненадежным. Что если бы существовал способ описания взаимодействий контроллеров представлений, который находился бы за пределами самих контроллеров, позволяя нам отслеживать весь поток данных и взаимодействий в одном месте?

Если вы задумывались над этими вопросами, то вы молодец! Оказывается, компания Apple тоже подумала об этом. Она включила в комплект iOS 5 SDK новую систему под названием **Storyboards** (Раскадровка — Примеч. ред.), предназначенную для решения этих проблем.

Система Storyboards основана на знакомой концепции nib-файлов и редактируется точно так же с помощью программы Interface Builder в среде Xcode. Но, в отличие от самих nib-файлов, система Storyboards позволяет работать с несколькими представлениями, каждое из которых связано со своим собственным контроллером, в отдельном визуальном рабочем пространстве. Вы можете конфигурировать переходы между контроллерами представлений, а также конфигурировать табличное представление с помощью фиксированного множества заранее определенных ячеек. В этой главе мы исследуем некоторые из этих возможностей, чтобы вы лучше узнали, что такое раскадровка, увидели, чем раскадровки отличаются от nib-файлов, и поняли, как их использовать в своих приложениях.

ЗАМЕЧАНИЕ. Как вы сами убедитесь, Storyboards — впечатляющий механизм. Впрочем, важно помнить, что, по крайней мере пока, он функционирует только на устройствах, работающих под управлением операционной системы iOS 5 и ее более новых версий. По мере того как все больше людей будут переходить к системе iOS 5, эта проблема станет менее важной.

Создание простой раскадровки

Начнем с простого проекта, демонстрирующего основные характеристики раскадровки. Находясь в среде Xcode, выберите команду File⇨New⇨New Project… для создания нового проекта. Выберите пиктограмму `Single View Application` в разделе iOS Application group и щелкните на кнопке Next. Назовите проект `Simple Storyboard`, установите флажок Use Storyboard и снова щелкните на кнопке Next. Выберите каталог, в котором хотите хранить свой проект, и щелкните на кнопке Create.

Как только проект будет создан, взгляните на навигатор проекта. Вы увидите хорошо знакомые файлы классов `BIDAppDelegate` и `BIDViewController`. Вы также увидите, что в нем нет класса `BIDViewController.xib`, но есть файл `MainStoryboard.storyboard`. Имя файла раскадровок не совпадает с именем контроллера представления, в отличие от nib-файла, поскольку раскадровка предназначена для удобного представления сразу нескольких контроллеров представлений.

Выберите файл `MainStoryboard.storyboard`, и программа Xcode откроет знакомое окно программы Interface Builder, показанное на рис. 10.1. Впрочем, между окном для редактирования nib-файлов, в котором мы работали до сих пор, и окном для редактирования раскадровок существует небольшая разница. Например, редактор раскадровок в программе Interface Builder не предусматривает режим значков при работе с раскадровками. Вместо этого после щелчка на треугольнике раскрытия, расположенном справа внизу от дока, весь док сворачивается и исчезает.

Другой пример связан с пиктограммами первого реагирующего объекта и контроллера представления. Если выбрать команду View Controller в доке, то пиктограммы First Responder и View Controller появятся не только в доке, но и под контроллером (рис. 10.2). В раскадровке каждое представление и его соответствующий контроллер всегда появляются вместе, образуя **сцену** (scene).

Кроме того, на представление, показанное в области редактирования, направлена большая стрелка. Этот элемент окажется удобным, когда мы немного позже будем создавать раскадровки, содержащие несколько представлений. Стрелка указывает на контроллер начального представления, которое должно загружаться и выводиться на экран при загрузке приложением данной раскадровки. Если раскадровка действительно содержит несколько представлений, то можете просто перетащить стрелку, чтобы она показывала на правильный

контроллер начального представления. Пока наш проект содержит только одно представление. Если вы попытаетесь перетащить стрелку в другое место, то увидите, что она вернется в исходное место, как только вы отпустите кнопку мыши.

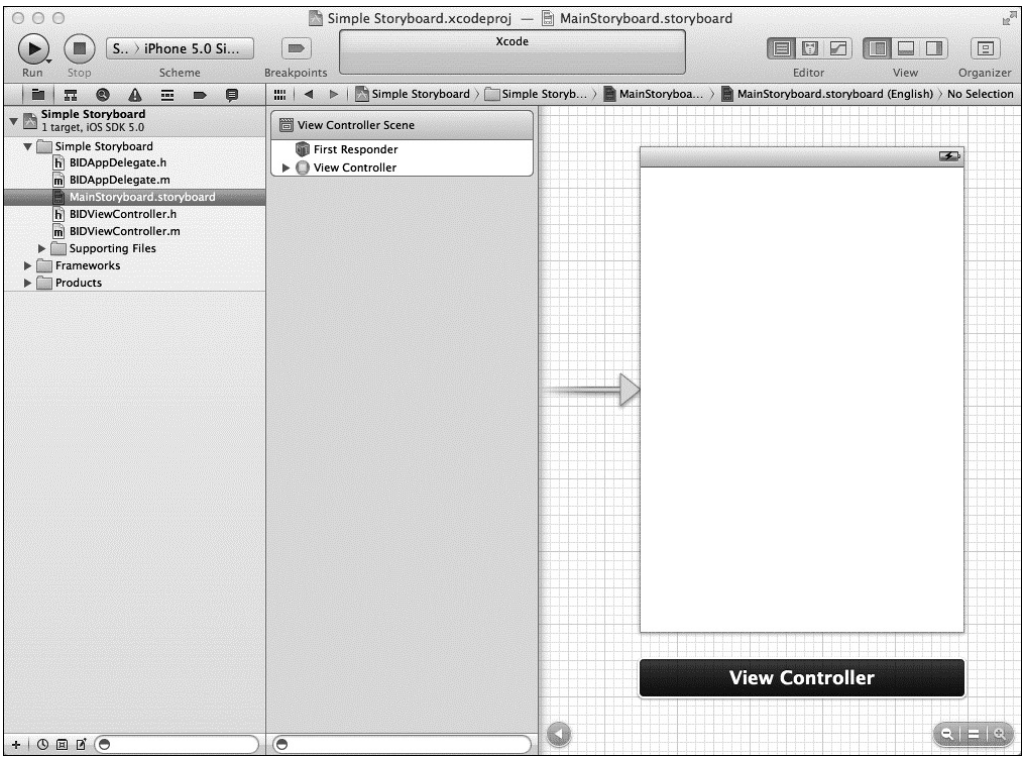

Рис. 10.1. Контроллер представления в раскадровке

Другое заметное изменение области редактирования заключается в возможности масштабирования набора элементов управления, расположенных в правом нижнем углу окна редактирования. Это удобно при работе с большим количеством контроллеров представления в раскадровке, поскольку позволяет видеть одновременно несколько контроллеров представления и связи между ними. Обратите внимание на то, что при уменьшении изображения программа Interface Builder не позволяет перетаскивать объекты из библиотеки объектов на свои представления. Кроме того, пользователь не может выбирать объекты в своих представлениях при уменьшении изображения. Итак, этот режим работы не очень удобен для редактирования представлений, но позволяет увидеть общую картину.

Добавим в представление метку. Увеличьте изображение и перетащите элемент Label из библиотеки объектов в центр представления. Дважды щелкните на метке, чтобы выбрать ее текст, и измените текст на `Simple`. Запустите приложение, и вы увидите на экране только что созданную метку.

До сих пор мы создавали представления на основе шаблонов, но при работе с раскадровками порядок действий немного изменяется. Посмотрим на оставшуюся часть нашего проекта, чтобы увидеть, что происходит "за кулисами" приложения, основанного на раскадровках.

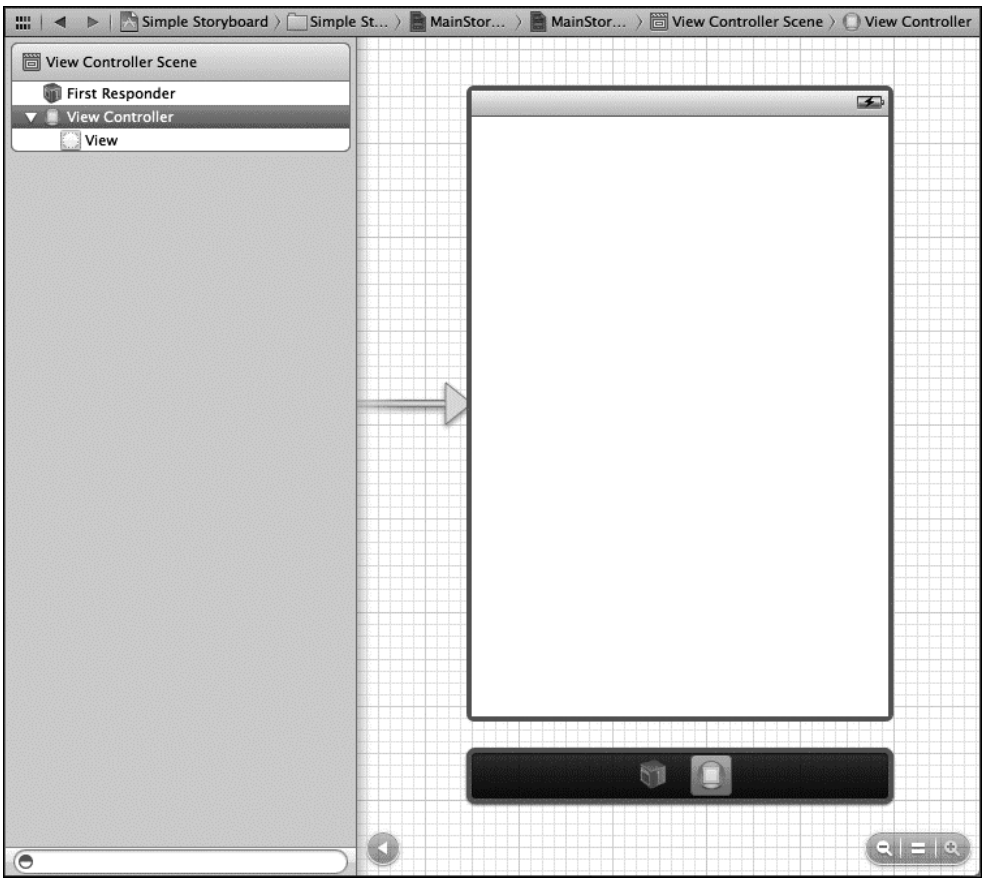

Рис. 10.2. Когда вы выбираете команду View Controller, под контроллером представления появляются пиктограммы First Responder и View Controller

Перейдите к навигатору проекта, выберите файл `BIDViewController.m` и просмотрите код. За исключением метода автоматического вращения все методы в этом файле посылают сообщение суперклассу, а затем возвращают значение. Разумеется, никакой речи о раскадровке тут не идет.

Перейдите к файлу `BIDAppDelegate.m`. Вы увидите ряд пустых методов. Обратите внимание на метод `application:didFinishLaunchingWithArguments:`, который выглядит иначе, чем метод с таким же именем, реализованный для других приложений. В приложениях, разработанных нами ранее, этот метод содержал код, создававший объект класса `UIWindow`, иногда открывал nib-файл и делал что-нибудь еще. На этот раз он совершенно пустой! Как же наше приложение догадается загрузить раскадровку и как начальное представление будет выведено на экран? Ответ на этот вопрос кроется в целевых настройках.

Перейдите к навигатору проекта и выберите самый верхний элемент Simple Storyboard, представляющий сам проект. Выберите также цель Simple Storyboard и вкладку Summary в верхней части экрана. Загляните в раздел iPhone / iPod Deployment Info, в котором вы увидите раскадровку `MainStoryboard`, сконфигурированную как `Main Storyboard` (рис. 10.3).

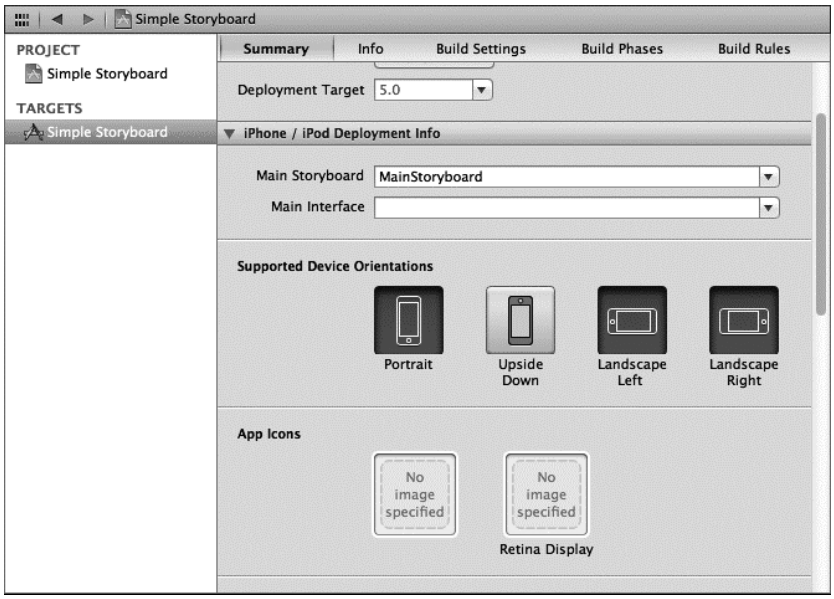

Рис. 10.3. Резюме цели проекта `Simple Storyboard`

Оказывается, вся конфигурация, которая необходима для того, чтобы ваше приложение автоматически создало окно, загрузила раскадровку и ее начальное представление, создало контроллер начального представления в раскадровке и смонтировало все это. Помимо прочего, это означает, что делегат приложения становится проще, поскольку создание окна и начального представления выполняется автоматически. Все "провода" остаются "за кулисами", и если вы включаете раскадровку в проект, то все упрощается.

Прототипы динамических ячеек

Как указывалось в главе 8, система iOS 5 позволяет создавать nib-файлы, содержащие класс `UITableViewCell`, и хранить в ячейке любые объекты, а также использовать уникальный идентификатор для регистрации этой ячейки в табличном представлении. Затем, на этапе выполнения, вы можете попросить табличное представление предоставить вам ячейку по ее идентификатору, и если заданный идентификатор совпадает с идентификатором, установленным при предыдущей регистрации, вы получите именно ее.

При использовании раскадровок эта концепция получает новое развитие. Теперь вместо создания отдельных nib-файлов для каждого типа ячеек вы можете создать их всех в одной раскадровке, прямо в табличном представлении, где будут представлены ячейки! Давайте посмотрим, как это делается.

Содержимое динамической таблицы в стиле раскадровок

Мы собираемся создать контроллер, выводящий на экран список элементов. В зависимости от содержимого каждого элемента будем выводить их на экран в обычном или экстравагантном стиле, чтобы предупредить пользователя о том, что он должен обратить особое

внимание на данный элемент. Для того чтобы конкретизировать сказанное, предположим, что эти элементы представляют собой записи в списке заданий, и мы хотим предупредить пользователей об опасном содержимом. Для простоты будем использовать обычный класс `UITableViewCell` для каждой ячейки, выводимой на экран, вместо определения подклассов для всех ячеек. В реальных, более сложных приложениях, вероятно, вам все же придется создавать для ячеек свои собственные подклассы. В любом случае настройки и рабочий процесс будут одинаковыми.

Вернемся к нашему проекту `Simple Storyboard`, раз уж мы его создали. Создайте новый класс в окне проектов Xcode. Выберите команду File⇨New⇨New File…, затем в разделе Cocoa Touch выберите пункт UIViewController subclass и щелкните на кнопке Next. Назовите класс `BIDTaskListController` и выберите класс `UITableViewController` во всплывающем списке Subclass of. Сбросьте флажок для создания нового xib-файла, поскольку вместо него мы будем работать с раскадровкой.

После создания файлов класса переключитесь на папку `MainStoryboard.storyboard`, где мы будем хранить экземпляр нового контроллера (и, разумеется, соответствующее представление). Захватите элемент Table View Controller из библиотеки объектов и перетащите его в область редактирования, разместив ее справа от начального контроллера. Вы должны увидеть на экране примерно то, что изображено на рис. 10.4. Скорее всего, вы увидите предупреждение, сообщающее, что ячейки таблицы `Tprototype` должны иметь повторно используемые идентификаторы. Но это не должно вас беспокоить: мы скоро исправим эту ошибку.

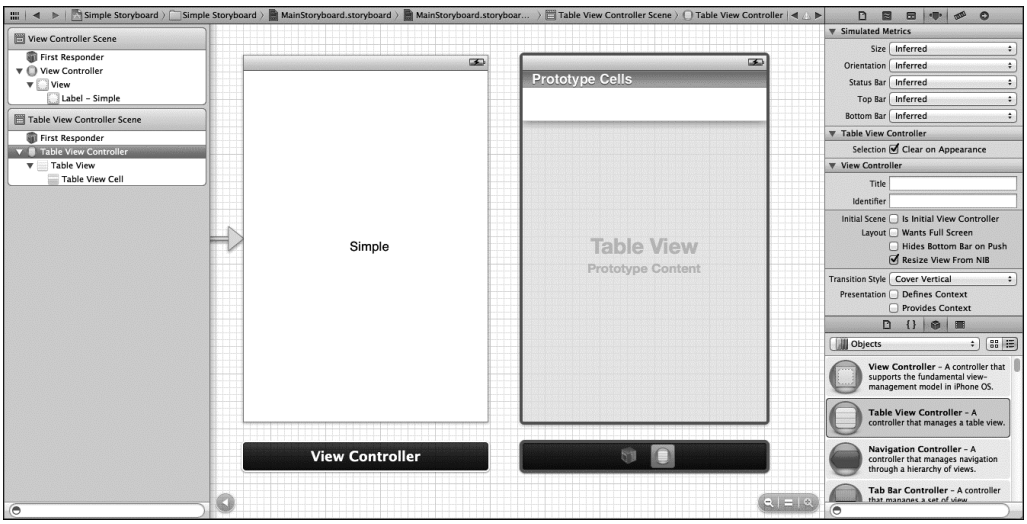

Рис. 10.4. Новый контроллер табличного представления, расположенный справа от оригинального контроллера представления

Теперь необходимо сконфигурировать контроллер табличного представления в качестве экземпляра класса нашего контроллера, а не стандартного класса `UITableViewController`. Выберите вновь вставленный контроллер табличного представления, следя при этом, чтобы выбран был только контроллер, а не табличное представление, которое в нем содержится. Для этого проще всего щелкнуть на панели пиктограмм, расположенной под табличным представлением, или на строке Table View Controller в доке. Вы увидите, что требуемые

элементы выбраны, когда и табличное представление, и панель пиктограмм будут обведены синим контуром. Откройте инспектор идентичности и измените класс контроллера на `BIDTaskListController`, чтобы табличное представление знало, где брать данные.

Редактирование клеток прототипа

Вы увидите, что табличное представление в верхней части с меткой Prototype Cells имеет группу, содержащую один элемент. Именно здесь можно графически располагать ячейки и присваивать им уникальные идентификаторы, чтобы ссылаться на них впоследствии в своем коде.

Начнем с выбора пустой ячейки, которая уже содержится в этой панели, и откроем инспектор атрибутов. Введите в поле Identifier строку `plainCell`, а затем перетащите элемент Label из библиотеки объектов прямо в ячейку. Перетащите метку, разместив ее над левым краем, чтобы появились голубые линии разметки, а затем измените ее ширину, перетаскивая правый край ячейки, пока голубые линии разметки не совпадут с правым краем ячейки. В заключение, выбрав метку, с помощью инспектора объектов задайте дескриптор равным `1` (рис. 10.5). Это позволит найти метку в коде.

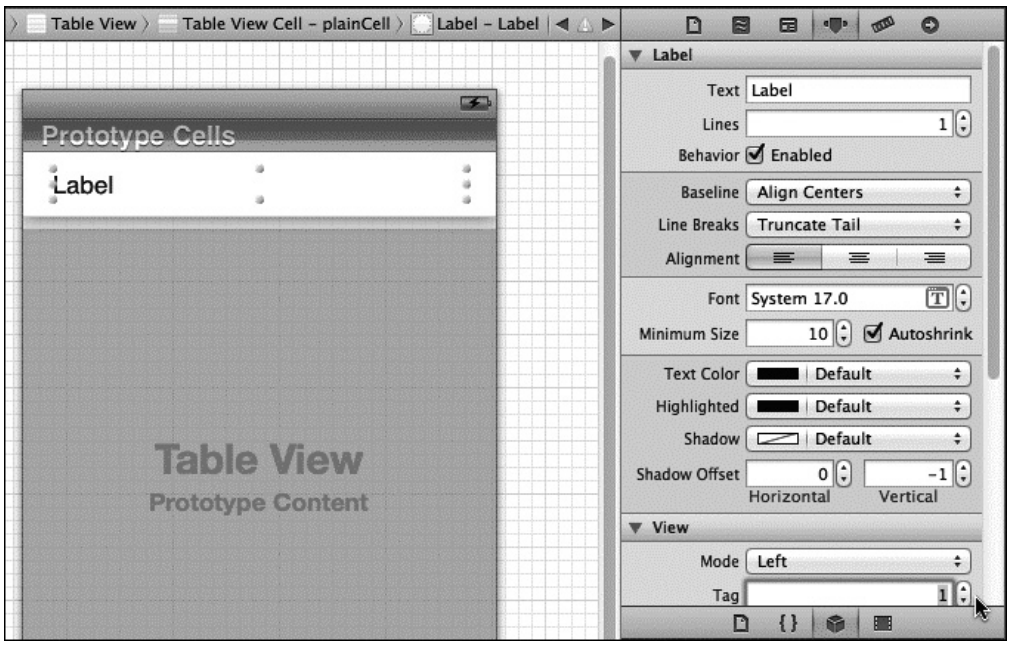

Рис. 10.5. Выбирая метку, мы используем инспектор атрибутов, чтобы изменить ее дескриптор на 1. Обратите внимание на то, что поле Tag находится в разделе View, в самом низу окна инспектора. На это поле указывает курсор

Теперь выберите саму ячейку табличного представления (но не метку, которую она содержит) и команду Edit⇨Duplicate. В результате под оригинальной ячейкой появится ее копия.

ЗАМЕЧАНИЕ. Выбор ячейки табличного представления может оказаться сложным делом. В версии программы Xcode, которую мы использовали при написании этой главы, нам пришлось щелкнуть на ячейке таблицы, чтобы продублировать ее. Выбрать ячейку в доке оказалось недостаточно. Возможно, со временем эта проблема исчезнет.

Выбрав новую ячейку, откройте инспектор объектов, чтобы установить идентификатор равным `attentionCell`. Затем выберите метку новой ячейки, с помощью инспектора атрибутов измените поле Text Color ячейки на `red` и задайте атрибут Font равным `System Bold`.

Итак, у нас есть две клетки прототипа, готовых к использованию в табличном представлении. Перед тем как создать код, заполняющий таблицу, нам необходимо внести еще одно изменение в раскадровку. Помните о большой плавающей стрелке, указывающей на оригинальное представление? Перетащите ее так, чтобы она указывала на новое представление. Сохраните раскадровку.

Источник данных для обычного табличного представления

Перейдите к файлу `BIDTaskListController.m`, в который мы собираемся добавить код, заполняющий таблицу. Это совершенно стандартный код для табличных представлений, который мы видели много раз, поэтому пропустим уже знакомые нам моменты и сосредоточимся на новшествах. Начнем с добавления строк, выделенных полужирным шрифтом, в начало файла.

```
#import "BIDTaskListController.h"

@interface()
@property (strong, nonatomic) NSArray *tasks;
@end

@implementation BIDTaskListController

@synthesize tasks;
```

Эти строки просто настраивают свойство, чтобы оно содержало список элементов, которые мы хотим демонстрировать на экране.

Вставьте следующий код в метод `viewDidLoad`, чтобы заполнить свойство заданий. Обратите внимание на то, что мы не приводим здесь комментарии.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.tasks = [NSArray arrayWithObjects:
                  @"Walk the dog",
                  @"URGENT:Buy milk",
                  @"Clean hidden lair",
                  @"Invent miniature dolphins",
                  @"Find new henchmen",
                  @"Get revenge on do-gooder heroes",
                  @"URGENT: Fold laundry",
                  @"Hold entire world hostage",
                  @"Manicure",
                  nil];
}
```

И, разумеется, мы должны соблюдать порядок и освобождать память, когда представление больше не выводится на экран.

```
- (void)viewDidUnload

{
   [super viewDidUnload];
```

```
    // Освобождаем все дочерние представления
    // главного представления, например self.myOutlet = nil;
    self.tasks = nil;

}
```

Теперь перейдем к самому контролеру и реализуем методы, наполняющие табличное представление содержанием. Начнем с простых методов, сообщающих табличному представлению, сколько в нем существует разделов и строк.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.
    // Возвращает количество разделов.
    return 0;
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.
    // Возвращает количество строк в разделе.
    return 0;
    return [tasks count];
}
```

Затем заменим содержимое метода, заполняющего каждую ячейку..

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
                             CellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                        reuseIdentifier:CellIdentifier];
    }

    NSString *identifier = nil;
    NSString *task = [self.tasks objectAtIndex:indexPath.row];
    NSRange urgentRange = [task rangeOfString:@"URGENT"];
    if (urgentRange.location == NSNotFound) {
        identifier = @"plainCell";
     } else {
        identifier = @"attentionCell";
     }

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:identifier];

    // Конфигурируем ячейку...

    UILabel *cellLabel = (UILabel *)[cell viewWithTag:1];
    cellLabel.text = task;

    return cell;
}
```

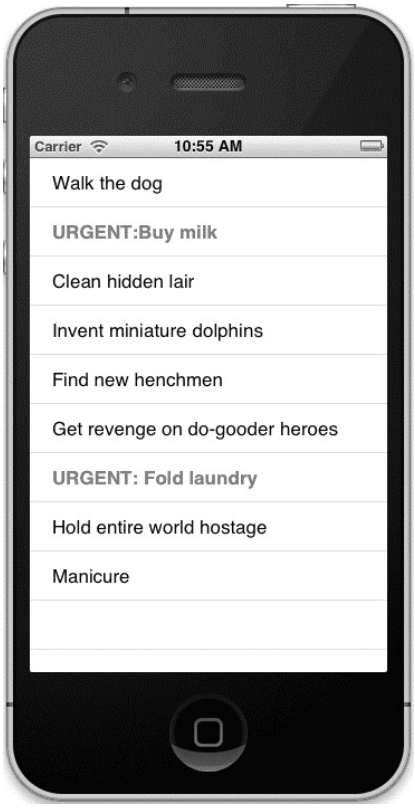

Рис. 10.6. Запуск приложения с ячейками раскадровки. Обратите внимание на два вида ячеек. Эту скрытую ловушку лучше выявить, не так ли?

Начинаем работу с того, что извлекаем задание из массива и проверяем, содержит ли оно слово "URGENT" ("СРОЧНО"). Это не очень сложный алгоритм поиска срочных заданий в списке текущих дел, но этого пока достаточно. Мы используем наличие или отсутствие данного слова, чтобы решить, какую ячейку хотим загрузить, и выяснить соответствующий идентификатор ячейки.

В главе 8 мы показали, как сообщить табличному представлению, что оно может найти ячейку по заданному идентификатору в nib-файле с конкретным именем. Можно размещать прототипы динамических ячеек и в табличном представлении, и в раскадровке. Разница состоит в том, что во втором случае вам не придется писать ни одной строчки кода, чтобы сообщить табличному представлению об этих прототипах ячеек. Вместо этого любое табличное представление, загруженное из раскадровки, будет автоматически иметь доступ к связанным с ним прототипам ячеек. Как и при работе с зарегистрированными nib-файлами, при вызове метода `dequeueReusableCellWithIdentifier:` табличное представление будет создавать такие ячейки на лету, поэтому проверять возвращаемое значение на равенство значению `nil` не обязательно. Благодаря этому остальная часть кода становится проще.

Для того чтобы увидеть это действие, щелкните на кнопке Run (рис. 10.6). При запуске приложения на экран выводится весь список заданий, в котором срочные задания выделены красным цветом. Обратите внимание на то, что данный код ничего не знает и никак не управляет цветом или шрифтом текста в каждой ячейке! Он лишь предполагает, что каждая ячейка содержит экземпляр класса `UILabel` с дескриптором, равным 1. Таким образом, графический пользовательский интерфейс отделяется от кода контроллера и внешний вид ячеек табличного представления можно легко изменять без вмешательства в исходный код.

Будет ли это загружаться?

Вероятно, стоит упомянуть о том, что создание экземпляра класса `BIDTaskListController` и его соответствующего представления происходит совершенно иначе, чем вы себе представляете. До сих пор каждый раз, когда нам требовалось вывести на экран контроллер представления, мы его явным образом создавали в каком-нибудь месте исходного кода, потом выполняли загрузку его представления из nib-файла (за исключением контроллеров табличного представления, которые обычно создаются без nib-файлов, вынуждая создавать их представления с нуля).

В простейших приложениях мы создавали контроллер представления и загружали его nib-файл в делегат приложения. В других случаях мы создавали его в ответ на какое-нибудь

действие пользователя, а затем "заталкивали" в стек навигации. Однако в данном примере контроллер представления создается автоматически при запуске приложения, точно так же ,как экземпляр класса `BIDViewController`, генерируемый шаблоном. Это чудо происходит во время запуска приложения, но существуют программные способы, с помощью которых можно вытолкнуть любой контроллер представления из раскадровки при запуске приложения. Мы рассмотрим их в этой главе немного позднее.

Статические ячейки

Посмотрим на новую конфигурацию табличного представления, которую позволяет использовать раскадровка, — статические ячейки. До сих пор все ячейки табличных представлений, которые мы видели (в этой и предыдущих главах), создавались и заполнялись динамически в методах контроллера класса `UITableViewDataSource`. Это удобно для вывода на экран списков, размер которых изменяется во время выполнения приложения, но иногда вы точно знаете, что хотите вывести на экран. Если количество элементов и виды ячеек полностью известны заранее, реализация объекта `dataSource` становится неприятной задачей.

К счастью, раскадровка предоставляет альтернативу динамическим ячейкам — статические ячейки! Теперь вы можете определить набор ячеек в табличном представлении, и они будут отображаться в работающем приложении точно так же, как в программе Interface Builder. Обратите внимание на то, что эти ячейки являются статическими только в том смысле, что существование самих этих ячеек обеспечивается при каждом запуске приложения. Однако их содержимое может изменяться на лету. Фактически вы можете соединить с ними выходы, чтобы иметь к ним доступ и задавать их содержимое из контроллера. Итак, приступим!

Выберите папку `Simple Storyboard` в браузере проекта Xcode и команду File⇨New⇨New File…. В разделе iOS / Cocoa Touch окна помощника для создания новых файлов выберите пункт UIViewController subclass и щелкните на кнопке Next. Назовите новый класс `BIDStaticCellsController`, выберите пункт UITableViewController во всплывающем списке Subclass of, сбросьте флажок, означающий создание соответствующего XIB-файла, и создайте новые файлы. Позднее, после настройки графического пользовательского интерфейса, мы внесем в эту реализацию контроллеров небольшие изменения.

Откройте файл `MainStoryboard.storyboard` и вытолкните из библиотеки следующий экземпляр класса `UITableViewController`, добавив его к предыдущим двум, которые уже включены в раскадровку. Переместите большую стрелку — ту, которая указывает на контроллер начального представления, — чтобы она указывала на новый контроллер.

Если вас беспокоит тот факт, что в раскадровке хранятся три контролера представления, но на экране отображается только один, успокойтесь! При создании реального приложения раскадровку обычно не заполняют ненужными представлениями и контроллерами, а пока мы просто экспериментируем. Ниже в этой главе мы покажем, как использовать несколько контроллеров представления в одной раскадровке.

Итак, вернемся к статическим ячейкам.

Как обеспечить статичность

Выберите табличное представление, которое вы только что перетащили в контроллер, и откройте инспектор атрибутов. Щелкните на всплывающем списке, расположенном в самом верху экрана и имеющем метку Content, и выберите не Dynamic Prototypes, а Static

Рис. 10.7. Изменения в статических ячейках в новом табличном представлении

Cells. Это изменит функционирование данного табличного представления, несмотря на то, что его внешний вид останется прежним. Теперь, когда это табличное представление и ее контроллер будут загружены в раскадровку, все ячейки, которые вы добавите, будут создаваться в том порядке, который вы укажете.

Для того чтобы эта таблица немного отличалась от списка заданий, выберите во всплывающем списке Style пункт Grouped. В первый момент это табличное представление имеет только один раздел и все ее углы закруглены, что характерно для типичных сгруппированных табличных представлений. Выберите раздел (не одну из ячеек, а весь раздел), и инспектор объектов выведет на экран несколько элементов, которые вы можете настроить. Установите количество строк равным `2`, а заголовком сделайте строку `Silliest Clock Ever` (Простейшие часы), поскольку именно это лучше всего отражает суть нашего контроллера.

Выберите первую ячейку и с помощью инспектора атрибутов установите атрибут Style равным Left Detail. Это один и стандартных стилей для отображения ячеек табличного представления. Он подразумевает, что слева от ячейки будет размещаться метка дескриптора, а также более крупная метка, содержащая реальное значение, которое мы хотим вывести на экран. Выберите текст метки, расположенной слева, дважды щелкнув на нем, и замените его на строку `The Date`. Повторите эту процедуру для второй метки, изменив ее текст на `The Time` (рис. 10.7).

Мы собираемся создать пару выходов для соединения нашего контроллера непосредственно с метками деталей, чтобы мы могли задавать их значения в ходе выполнения программы. Выберите новый контроллер представления в доке, вызовите инспектор идентичности и измените поле Class контроллера представления на `BIDStaticCellsController`. Нажмите клавишу <Return>, чтобы зафиксировать изменения. Имя контроллера в доке должно измениться на Static Cells Controller.

Выберите пиктограмму Static Cells Controller в доке. Вызовите помощник редактора и откройте в нем файл `BIDStaticCellsController.h`.

Выберите правую метку в первой ячейке табличного представления (следующую за меткой The Date) и проведите от нее соединительную линию на заголовочный файл, отпустив кнопку мыши между строками `@interface` и `@end`. В появившемся всплывающем списке установите поле Name равным dateLabel, а остальные значения оставьте по умолчанию. Затем повторите эту процедуру для второй ячейки, но на этот раз назовите ее timeLabel. Этими простыми действиями мы создали свойства нового выхода и правильно соединили их всех одновременно.

Перейдите к файлу `BIDStaticCellsController.m`, чтобы вывести на экран дату и время.

Старый добрый источник данных для табличного представления

Открыв файл BIDStaticCellsController.m, мы должны удалить методы dataSource. Все три метода должны исчезнуть! В противном случае наше табличное представление не поймет, что именно оно должно выводить на экран. Просто полностью удалите эти методы, включая содержимое между фигурными скобками (которое мы здесь не показываем).

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    ...
}
- (NSInteger)tableView:(UITableView *)tableView
        numberOfRowsInSection:(NSInteger)section
{
    ...
}
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    ...
}
```

Удалив эти методы, вывести на экран дату и часы так же просто, как вставить следующие строки в конец метода viewDidLoad:

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

 // Некоторые комментарии пока можно игнорировать!

 NSDate *now = [NSDate date];
 dateLabel.text = [NSDateFormatter localizedStringFromDate:now
                                   dateStyle:NSDateFormatterLongStyle
                                   timeStyle:NSDateFormatterNoStyle];
 timeLabel.text = [NSDateFormatter localizedStringFromDate:now
                                   dateStyle:NSDateFormatterNoStyle
                                   timeStyle:NSDateFormatterLongStyle];
}
```

Здесь мы определяем текущую дату, а затем используем удобный класс NSDateFormatter, чтобы вывести их на экран по отдельности, передав разным меткам. Это все, что требовалось сделать! Щелкнув на кнопке Run, вы увидите наши великолепные, новые и превосходно работающие часы (рис. 10.8).

Как видим, использовать табличное представление со статическими ячейками оказалось проще, чем применять подход, основанный на использовании методов dataSource, поскольку количество ячеек в нашем проекте фиксировано. Если хотите вывести на экран более гибкий набор данных, следует использовать методы dataSource, но статические ячейки позволяют проще создавать определенные типы экранов, например меню и списки настроек.

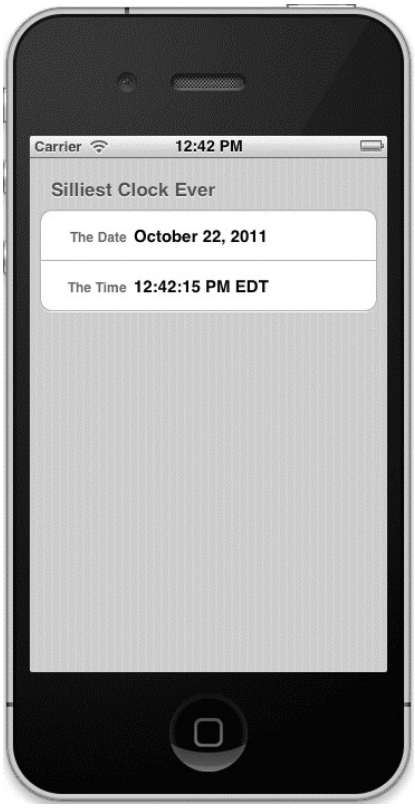

Рис. 10.8. Наши простейшие часы во всей красе

Вы говорите: "переход", я говорю: "переход"

Настало время перейти к следующей функциональной возможности, которую компания Apple встроила в систему iOS 5, — к **переходу** (segue). С помощью переходов мы можем использовать программу Interface Builder для того, чтобы определить, как одна сцена переходит в другую. Этот подход применим только к раскадровкам, но не к nib-файлам.

Идея заключается в том, что вы можете использовать одну раскадровку для множества графических пользовательских интерфейсов своих приложений со всеми сценами и их взаимными переходами, представленными в графическом макете. Этот подход имеет массу побочных эффектов. Один из них состоит в том, что вы можете видеть и редактировать весь поток вашего приложения в одном месте, что намного упрощает работу с большими приложениями. Кроме того, как вы вскоре убедитесь, это позволяет исключить часть кода из ваших контроллеров представлений.

ЗАМЕЧЕАНИЕ. Мы заметили некоторую несогласованность между словом segue и его произношением. В английском языке есть слово transition, заимствованное из итальянского, означающее переход. В основном оно используется в определенных журналистских и музыкальных контекстах и встречается редко. Слово seque произносится как "segway," точно так же, как транспортные устройства Segway (название которых, и это должно быть теперь очевидным, является искажением слова segue).

Переходы наиболее полезны в приложениях, основанных на механизмах навигации, которые мы рассматривали на протяжении последних нескольких глав. Здесь мы не собираемся создавать что-то грандиозное, подобное приложению из главы 9, но все же продемонстрируем, как можно использовать сочетание переходов и статических таблиц.

Создание навигатора переходов

Примените программу Xcode для создания нового приложения iOS, выбрав шаблон Empty Application. Назовите проект `Seg Nav`. В результате будет создано такое же приложение, которые вы уже видели в предыдущих главах, где был описан код для создания окна в делегате приложения, не относящийся к раскадровке. Поскольку мы собираемся создать раскадровку и хотим, чтобы она загружалась автоматически, как в предыдущем проекте, выберите файл `BIDAppDelegate.m`, найдите метод `application:didFinishLaunchingWithOptions:` и удалите из него все, кроме последней строки.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Точка замещения для настройки приложения после запуска.
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

Выберите папку `Seg Nav` в навигаторе проекта и команду File⇨New⇨New File… для создания нового файла. В разделе User Interface выберите пункт Storyboard и щелкните на кнопке Next. Назовите новый файл `MainStoryboard.storyboard`.

После создания файла необходимо сконфигурировать наш проект так, чтобы раскадровка загружалась при запуске приложения. Выберите пиктограмму проекта `Seg Nav` в верхней части навигатора проекта. На целевой вкладке Summary вы обнаружите кнопку всплывающего списка, которая позволит задать раскадровку Main Storyboard. Щелкните на всплывающем списке и выберите пункт MainStoryboard (в этом списке всего одно имя, так что ошибиться невозможно).

Заполнение пустой грифельной доски

Выберите файл `MainStoryboard.storyboard` в навигаторе проекта. Вы увидите обычную область для создания макета, но на этот раз совершенно пустую. Найдите в библиотеке объектов элемент Navigation Controller и перетащите его на область макетирования, чтобы создать начальную сцену.

Напомним, что класс `UINavigationController` сам не отображает никакого содержимого. Он просто выводит на экран панель навигации. Итак, когда вы перетаскиваете объект класса `UINavigationController` в свою раскадровку, программа Interface Builder в тот же момент создает объект класса `UIViewController` и его представление. Вы увидите два контроллера, расположенных рядом, и особую стрелку, направленную от контроллера навигации к контроллеру представления. Эта стрелка отображает свойство контроллера навигации `rootViewController` и связана с контроллером представления, так что вы должны что-нибудь загрузить.

Начнем заполнять это представление определенным содержимым, но в данном случае было бы поучительным создать приложение, основанное на раскадровке, которое имитировало бы

поведение приложения `Nav`, разработанное в главе 9. Итак, в качестве стартового экрана вставим контроллер табличного представления. Для этого выберите контроллер представления, расположенный справа (тот, на который показывает стрелка, а не тот, откуда она исходит), и нажмите клавишу <Delete>, чтобы удалить его. Затем перетащите элемент Table View Controller из библиотеки объектов на область макетирования и разместите его справа от контроллера навигации, там, где находился удаленный контроллер представления.

В данный момент контроллер навигации не знает, где находится его объект класса `rootViewController`, поэтому мы должны восстановить связь с новым контроллером представления. Нажмите клавишу <Ctrl> и перетащите курсор от контроллера навигации к контроллеру табличного представления, и вы увидите всплывающий список, который выглядит точно так же, как и всплывающий список, позволяющий соединить выходы и действия (рис. 10.9). Однако на этот раз вместо выходов и действий этот список содержит группу соединений под названием Storyboard Segues. В этом списке есть пункт Relationship - rootViewController, за которым следуют три пункта: Push, Modal и Custom. Последние три пункта используются для создания переходов между сценами, и мы их вскоре исследуем. Пока выберите пункт rootViewController, чтобы установить это соединение.

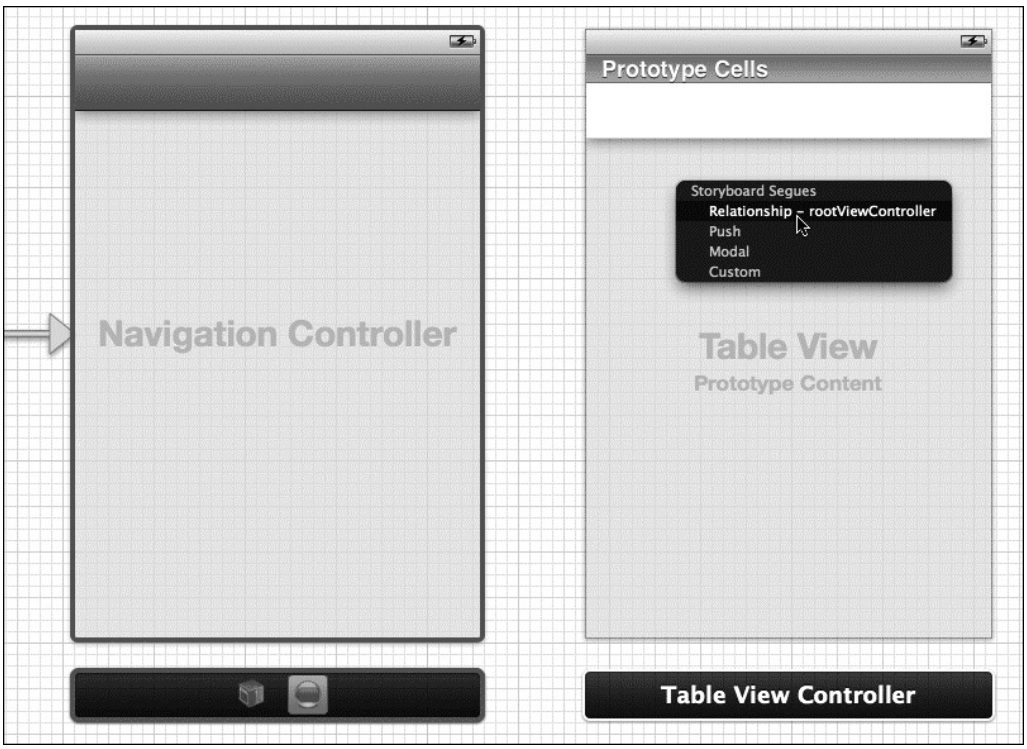

Рис. 10.9. Это изображение демонстрирует всплывающее меню, которое появляется в результате перетаскивания контроллера навигации на новый контроллер табличного представления при нажатой клавише <Ctrl>. Мы выбрали элемент, который задает контроллер табличного представления, как корневой контроллер представления

Теперь сделаем так, чтобы это табличное представление выводило на экран меню. Мы хотим, чтобы оно работало так же, как и корневое табличное представление в приложении `Nav` из главы 9. Однако со статическими ячейками таблицы все становится намного проще!

Выберите в доке табличное представление (но не контроллер табличного представления) и с помощью инспектора атрибутов измените ее атрибут Content на `Static Cells`. Вы увидите, что табличное представление немедленно получит три ячейки. Мы будем использовать только две из них, поэтому выберите одну ячейку и удалите ее.

Выберите каждую из двух оставшихся ячеек табличного представления и с помощью инспектора измените ее атрибут Style на `Basic`, чтобы ячейки получили заголовки. Затем измените эти заголовки на `Single view` и `Sub-menu` соответственно (рис. 10.10).

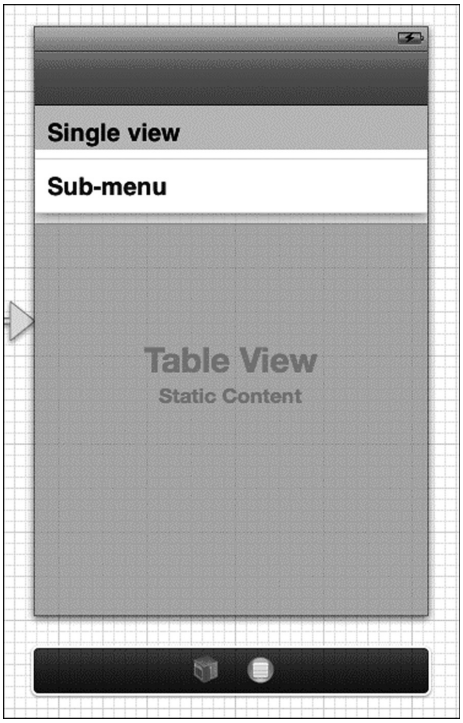

Рис. 10.10. Две статические ячейки с измененными заголовками

Зададим несколько заголовков, чтобы табличное представление хорошо взаимодействовало с контроллером навигации. Выберите элемент навигации (он выглядит как пустая панель инструментов), расположенный в верхней части табличного представления. Используя инспектор атрибутов, задайте атрибут Title равным `Segue Navigator`, а атрибут Back Button — `Seg Nav`. Обратите внимание на то, что атрибут Back Button не определяет значение, которое будет выводиться на экран вместе с представлением; вместо этого он определяет значение, которое следующий контроллер представления будет выводить на его кнопке возврата, ведущей к корневому представлению.

Запустите приложение, чтобы понять, как взаимодействуют его элементы. Вы должны увидеть только что созданное корневое табличное представление, содержащее две ячейки и заголовки.

Это все, что требуется для данного корневого представления, и все это мы сделали в программе Interface Builder, не написав ни одной строчки кода! Вы можете напомнить, что в главе 9 в приложении `Nav` у нас был подкласс `UITableViewController`, содержавший код для размещения и инициализации всех других контроллеров представления. Каждый раз, когда мы добавляли новый вспомогательный контроллер, мы должны были импортировать его заголовочный файл и добавлять код для создания его экземпляра. Благодаря раскадровкам теперь ничего этого делать не надо.

Добавляя вспомогательный контроллер в раскадровку, мы соединим ячейку корневого табличного представления с переходом в программе Interface Builder. Наш класс корневого контроллера не обязан знать о других контроллерах, поскольку он не связан с их созданием или выводом на экран. Вот почему мы работали с самим классом `UITableViewController`, не создавая его подклассов.

Первый переход

Настало время создать наш первый переход. Сначала перетащите элемент View Controller из библиотеки объектов в область макетирования, оставив его справа от остальных представлений. Мы не планируем отображать здесь какое-либо специальное содержимое, потому

что просто хотим войти в переход. Иначе говоря, нам нужен какой-то способ визуального подтверждения, что на экране отображается правильное представление, поэтому перетащите объект класса `UILabel` из библиотеки объектов на новое представление, измените его текст на Single View и центрируйте метку относительно представления по вертикали и по горизонтали.

Теперь начинается чудо создания перехода. Нажмите клавишу <Ctrl> и перетащите курсор с ячейки Single view в контроллере табличного представления на только что созданное представление. Вы снова увидите всплывающий список Storyboard Segues, но на сей раз без отношения `rootViewController`, которое мы видели раньше. Этот список содержит только три вида переходов, поддерживаемых программой Interface Builder: Push, Modal и Custom. Поскольку мы хотим реализовать модель стандартного контроллера навигации, предусматривающего заталкивание контроллеров в стек, выберите пункт Push. После этого на экране появится новая стрелка, символизирующая соединение, которая указывает на новый контроллер и исходит из корневого контроллера представления. Кроме того, в строке Single view в корневом табличном представлении справа появится стрелка раскрытия, сообщающая пользователю, что, щелкнув на ячейке, он может вывести на экран дополнительную информацию.

Еще раз запустите приложение, и вы увидите, что верхняя ячейка теперь имеет стрелку раскрытия, касание которой означает переход к новому представлению, которое мы только что настроили. Вы увидите добавленную нами метку, заголовок в верхней части экрана и кнопку возврата, которая называется Seg Nav (мы ее настраивали пару страниц назад). Нажмите клавишу возврата и вернитесь к корневому представлению.

Итак, мы создали большую часть представления навигации, не написав ни строчки кода. Это одно из основных преимуществ, которыми обладают переходы. Однако, разумеется, мы не можем не писать код вообще. Попробуйте создать графический пользовательский интерфейс, который выводит на экран список в виде обычного динамически масштабируемого табличного представления, так, чтобы выбор одного из его элементов открывал новый список настроек. Обратите внимание на то, что для перехода от выбранного элемента к контроллеру настроек необходимо написать довольно большой фрагмент кода. В то же время с помощью переходов это можно сделать много проще.

Немного более полезный список заданий

Мы собираемся написать немного более содержательную версию списка заданий, созданного в примере `Simple Storyboard`. В нашей версии вы увидите список заданий и сможете нажимать на них, чтобы редактировать. К счастью, это несложно.

Откройте проект `Simple Storyboard`, созданный нами ранее, и перетащите файлы `BIDTaskListController.h` и `BIDTaskListController.m` из этого проекта в папку `Seg Nav`, принадлежащую проекту `Seg Nav`. Установите флажок, заставляющий программу Xcode копировать файлы в проект. В этом случае файлы из папки проекта `Simple Storyboard` будут скопированы в папку `Seq Nav`, и к ним будут добавлены два файла из проекта `Seq Nav`.

Выберите папку `Seg Nav` в навигаторе проекта и команду File⇨New⇨New File…, чтобы создать новый файл. Выберите раздел Cocoa Touch, расположенный слева, затем пункт UIViewController subclass и щелкните на кнопке Next. Назовите этот класс `BIDTaskDetailController` и выберите в качестве суперкласса класс `UIViewController`. При этом программа Xcode не создает сопутствующий nib-файл. Создайте этот класс, но пока не редактируйте его. Настроив графический пользовательский интерфейс, мы еще вернемся к нему.

Переключитесь на файл `MainStoryboard.storyboard`. На этот раз мы создадим следующую сцену в нашей раскадровке, которая будет представлять собой список заданий. Перетащите объект класса `UITableViewController` из библиотеки объектов на область

макетирования, поместив его справа от остальных контроллеров. С помощью инспектора идентичности измените имя нового контроллера класса на `BIDTaskListController`.

Зайдите в док и выберите табличное представление, связанное с новым контроллером. Затем откройте инспектор атрибутов и зайдите в меню Content. Поскольку добавленное нами табличное представление будет отображать на экране динамический список элементов, оставим атрибут Dynamic Prototypes, а не Static Cells.

Мы будем использовать те же два прототипа ячеек, которые были описаны ранее в разделе "Прототипы динамических ячеек". Для того чтобы настроить эти прототипы, вернитесь к этому разделу либо откройте проект `Simple Storyboard` и скопируйте оттуда обе ячейки, вставив их в новую таблицу. Напомним, что каждая ячейка должна иметь метку, дескриптор, равный 1, а также идентификаторы `plainCell` и `attentionCell`.

Если хотите скопировать две ячейки из проекта `Simple Storyboard`, а не создавать их заново, перейдите к проекту `Simple Storyboard` и найдите две ячейки табличного представления в сцене `Task List Controller Scene`. Они должны называться `plainCell` и `attentionCell`. Закройте треугольники раскрытия и выберите обе ячейки (таким образом вы получите и метки). Скопируйте их, а затем вернитесь к проекту `Seg Nav`, выберите новое табличное представление и вставьте копию. Ваши ячейки и метки должны появиться в новом контроллере. Удалите исходную ячейку, оставив два прототипа клеток вместо трех.

Просмотр деталей задания

Добавим еще одну сцену, чтобы управлять отображением деталей, которые будут выводиться на экран, когда пользователь выберет строку в списке заданий. Перетащите из библиотеки объектов в область макетирования объект класса View Controller (но не класса Table View Controller), поместив его справа от предыдущей сцены. Используя инспектор идентичности, измените класс контроллера на `BIDTaskDetailController`.

Эта сцена посвящена редактированию деталей выбранного задания, которое в простом случае означает редактирование строки. Для этого будем использовать класс `UITextView` — полнофункциональный многострочный текстовый редактор, встроенный в систему iOS. Перетащите объект этого класса из библиотеки объектов в новую сцену. Вы увидите, что он заполнит собой все пространство. Поскольку мы хотим предоставить пользователям возможность редактировать текст, такое положение вещей нас не устроит. Мы предпочитаем работать с маленьким текстовым представлением, которое заполняет на экране место, свободное от клавиатуры. Захватите маркер изменения размера в центре нижнего края и перетащите его вверх, пока высота текстового представления не составит примерно 200 пикселей. Когда начнете перетаскивать представление, в его верхней части появится координата, которая облегчит выполнение вашего задания.

Затем необходимо добавить выход в контроллер, чтобы он мог найти текстовое представление и извлечь содержащуюся в нем строку. Мы можем создать и соединить этот выход, используя технологию "перетащить в код", встроенную в программу Interface Builder. Сначала откроем помощник редактора. Если вы выбрали класс `BIDTaskDetailController` или любую часть его представления в редакторе макета, помощник редактора должен показать заголовочный файл контроллера `BIDTaskDetailController.h`. Если нет, значит, помощник редактора, возможно, работает не в автоматическом режиме. Эту ситуацию можно исправить, используя панель быстрых переходов, расположенную над помощником редактора, щелкнув на пиктограмме, находящейся справа от стрелок, и выбрав команду Automatic.

Теперь выберите текстовое представление и, нажав клавишу <Ctrl>, перетащите его в код, оставив где-то между объявлениями `@interface` и `@end`. В появившемся всплывающем

меню установите тип Connection равным Outlet и задайте имя textView. Кроме того, установите атрибут Storage равным Weak и щелкните на кнопке Connect. В результате в файле BIDTaskDetailController.h возникнет новый выход, а в файле BIDTaskDetailController.m будут синтезированы новый get- и set-методы.

Еще несколько переходов, пожалуйста

Настроив графический пользовательский интерфейс для новых сцен, соединим их с помощью переходов. Вернитесь к корневому табличному представлению (которое содержит ячейки Simple View и Sub-menu). Выберите ячейку Sub-menu, нажмите клавишу <Ctrl> и перетащите курсор от этой ячейки к контроллеру списка заданий. Вероятно, легче всего это сделать в доке. Обратите внимание на то, что при этом мы пропускаем сцену Single View. В появившемся всплывающем списке выберите пункт Push, чтобы переход затолкнул сцену списка задания в стек, когда пользователь нажмет ячейку Sub-menu.

Выберите первый прототип ячейки в сцене контроллера списка заданий (он называется Table View Cell – plainCell и находится в доке; возможно, чтобы его увидеть, надо щелкнуть на треугольнике раскрытия). Нажмите клавишу <Ctrl> и перетащите курсор от нее к контроллеру Task Detail Controller, снова выбрав пункт Push. Сделайте то же самое со вторым прототипом ячейки (он называется Table View Cell – attentionCell и находится в доке), так, чтобы оба прототипа были соединены с контроллером Task Detail Controller. На экране появятся две стрелки, проведенные от списка заданий к деталям задания. Каждая стрелка представляет собой переход и каждая исходит из конкретного прототипа ячейки. Когда позднее вы запустите приложение, все ячейки, созданные для списка заданий, будут скопированы из одного из этих прототипов, поэтому каждый из них должен иметь переход, позволяющий им создавать и активизировать контроллер деталей задания.

На данный момент разработка графического пользовательского интерфейса завершена. Нам осталось только реализовать несколько методов для передачи выбранного задания из представления списка к детализированному представлению, а затем проложить другой путь, чтобы пользователь мог редактировать задания.

Передача задания из списка

Выберите файл BIDTaskListController.m и добавьте в класс, находящийся в конце файла, непосредственно над объявлением @end, следующий метод:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    UIViewController *destination = segue.destinationViewController;
    if ([destination respondsToSelector:@selector(setDelegate:)]) {
        [destination setValue:self forKey:@"delegate"];
    }
    if ([destination respondsToSelector:@selector(setSelection:)]) {
       // подготовка информации о выборе
       NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];
       id object = [self.tasks objectAtIndex:indexPath.row];
       NSDictionary *selection = [NSDictionary dictionaryWithObjectsAndKeys:
                                  indexPath, @"indexPath",
                                  object, @"object",
                                  nil];
       [destination setValue:selection forKey:@"selection"];
    }
}
```

Новый метод `prepareForSegue:sender:` вызывается в нашем контроллере, если представление контроллера было заменено другим представлением контроллера в результате активизации перехода. В данном случае это означает следующее: когда пользователь выберет любую ячейку в нашем табличном представлении, она активизирует связанный с ней переход, и этот метод будет вызван в нашем контроллере. Это дает нам возможность подготовить данные для передачи следующему контроллеру. Раньше эту задачу мы решали в методе делегата табличного представления, который вызывался при выборе строки, но новый способ является более гибким. Например, мы можем заменить наши ячейки табличного представления кнопками, и поскольку эти кнопки используют переходы для запуска других контроллеров представления, и этот метод будет вызван точно так же, как и сейчас.

Нам осталось сделать еще несколько модификаций, поэтому изучим метод `prepareForSegue:sender:`. С помощью параметра, который он получает, мы имеем доступ к контроллеру `destinationViewController` (который должен быть выведен на экран в данный момент) и контроллеру `sourceViewController` (который должен быть удален с экрана). В данном случае мы используем свойство `destinationViewController`, чтобы конфигурировать контроллер детализированного представления, поэтому поместим его в локальную переменную.

```
UIViewController *destination = segue.destinationViewController;
```

Затем сконфигурируем делегата целевого контроллера так, чтобы он ссылался на точку вызова. Это позволит целевому методу возвращать данные по завершении работы. В данный момент наш экран детализированного представления не имеет свойства делегата, но позднее будет иметь.

```
if ([destination respondsToSelector:@selector(setDelegate:)]) {
 [destination setValue:self forKey:@"delegate"];
 }
```

Кодирование "ключ–значение"

Вместо вызова метода `setDelegate:` мы используем кодирование "ключ–значение" (key-value coding — KVC), позволяющее косвенно применять get- и set-методы к любому объекту, задавая строки вместо имен методов. Механизм KVC является одной из основных функциональных возможностей каркасов Cocoa Touch, а его главные методы — `setValue:forKey:` и `valueForKey:` — встроены в класс `NSObject` и поэтому доступны отовсюду. Мы не рассматриваем этот вопрос подробно, но вскоре будем использовать данный механизм.

Почему мы используем механизм KVC вместо непосредственной настройки делегата? Одно из преимуществ механизма KVC заключается в том, что он освобождает нас от необходимости знать специфику интерфейсов других классов, что приводит к менее тесно связанному коду. Если мы хотим вызвать метод непосредственно, то должны объявить его интерфейс, содержащий метод `setDelegate:`, и преобразовать тип целевой переменной в тип, реализующий этот метод. Благодаря механизму KVC наш код не обязан знать ничего о методе `setDelegate:` (кроме факта, что получатель отвечает на него), поэтому нам не нужно объявлять для него никакого интерфейса. Класс `BIDTaskListController` не импортирует заголовочный файл класса `BIDTaskDetailController` и фактически даже ничего не знает о его существовании, и это очень хорошо! В общем, чем меньше зависимостей между классами, тем лучше. При условии правильного применения механизм KVC может помочь нам добиться указанной цели.

Теперь поместим выбранное задание и индекс выбранной строки в словарь, чтобы передать их контроллеру детализированного представления. Мы должны включить в словарь индекс строки, чтобы впоследствии, когда детализированное представление будет выведено на экран, контроллер мог передать нам обратно этот индекс, сообщая, какое задание было изменено. В противном случае мы могли бы просто передавать обратно строку и не знать, в каком месте списка она должна находиться.

```
if ([destination respondsToSelector:@selector(setSelection:)]) {
   // подготовка информации о выборе
   NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];
   id object = [self.tasks objectAtIndex:indexPath.row];
   NSDictionary *selection = [NSDictionary dictionaryWithObjectsAndKeys:
                              indexPath, @"indexPath",
                              object, @"object",
                              nil];
   [destination setValue:selection forKey:@"selection"];
}
```

Как и при настройке целевого делегата, задаем его выбор с помощью механизма KVC. В данный момент наш контроллер детализированного представления не имеет свойства для выбора, но мы уже можем его заменять.

Обработка деталей заданий

Выберите файл `BIDTaskDetailViewController.h`. Вы увидите свойство `textView`, которое было добавлено, когда мы перетащили выход из раскадровки. Добавьте еще два свойства.

```
#import <UIKit/UIKit.h>

@interface BIDTaskDetailController : UIViewController
@property (weak, nonatomic) IBOutlet UITextView *textView;
@property (copy, nonatomic) NSDictionary *selection;
@property (weak, nonatomic) id delegate;
@end
```

Обратите внимание на то, что свойство выбора задает хранилище копии (которым обычно является хранилище, используемое при работе с классами, содержащими значения, например `NSString` и `NSDictionary`), а делегат определяет слабое хранилище (weak storage). Нам необходимо использовать слабое хранилище для свойства делегата, чтобы случайно не забыть в памяти делегата, который, в свою очередь, забыл о нас! В данном случае нам известно, что делегат не хранит объект, но в стандартном шаблоне, используемом в каркасах Cocoa Touch, это следует гарантировать, поэтому у нас нет причин поступать как-то иначе.

Переключитесь на файл `BIDTaskDetailViewController.m` и добавьте в его начало код для синтезирования get- и set-методов для новых свойств.

```
@implementation BIDTaskDetailController
@synthesize textView;
@synthesize selection;
@synthesize delegate;
.
.
.
```

Прокрутите файл вниз, и вы увидите, что метод `viewDidLoad` по умолчанию закомментирован. Удалите символы комментариев и вставьте в это место следующий код:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    textView.text = [selection objectForKey:@"object"];
    [textView becomeFirstResponder];
}
```

К моменту вызова этого кода переход уже будет в стадии реализации, и контроллер представления списка уже настроит свое свойство выбора. Мы вытолкнем из стека значение, которое он содержит, и передадим его текстовому представлению. Затем сообщим текстовому представлению, чтобы оно стало первым реагирующим элементом, и на экране немедленно появится виртуальная клавиатура.

Запустите это приложение, пройдите до списка заданий и выберите задание. Вы должны увидеть значение, которое появится в поле редактирования. Пока все в порядке.

Передача деталей обратно

Осталось вернуть результаты редактирования пользователя обратно в список. К сожалению, наше детализированное представление не может вызвать метод `prepareForSegue:sender:`, когда пользователь нажимает клавишу возврата. Этот метод вызывается, только когда переход заталкивает новый контроллер в стек, а не когда выталкивает контроллер из него. Вместо этого мы будем использовать стандартный метод класса `UIViewController`, предназначенный для выполнения примерно таких же заданий, которые мы выполняли в классе `BIDTaskListController`. Добавьте этот метод после метода `viewDidLoad`.

```
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];

    if ([delegate respondsToSelector:@selector(setEditedSelection:)]) {
        // конец редактирования
        [textView endEditing:YES];
        // подготовка информации о выборе
        NSIndexPath *indexPath = [selection objectForKey:@"indexPath"];
        id object = textView.text;
        NSDictionary *editedSelection = [NSDictionary dictionaryWithObjectsAndKeys:
                                         indexPath, @"indexPath",
                                         object, @"object",
                                         nil];
        [delegate setValue:editedSelection forKey:@"editedSelection"];
    }
}
```

Здесь мы настраиваем наше свойство делегата `editedSelection` с помощью механизма KVC. Как и раньше, это освобождает нас от необходимости знать что-либо конкретное о классе другого контроллера. Единственный фрагмент, который может показаться вам новым, выглядит так: `[textView endEditing:YES]`. Он просто вынуждает текстовое представление завершить любое редактирование пользователя, чтобы обновить текстовое значение (которое мы извлечем через несколько строк).

Как заставить список получать детали

Последний фрагмент нашей мозаики готов к реализации. Мы должны вернуться к контроллеру представления списка, чтобы убедиться, что он может получать свойство

editedSelection и делать с ним нечто разумное. Вернитесь к файлу BIDTaskListController.m и внесите в расширение класса, находящееся в начале файла, следующие изменения:

```
@interface BIDTaskListController ()
~~@property (strong, nonatomic) NSArray *tasks;~~
@property (strong, nonatomic) NSMutableArray *tasks;
@property (copy, nonatomic) NSDictionary *editedSelection;
@end
.
.
.
```

Первое изменение заставляет включить свойство заданий в изменяемый массив, чтобы мы могли изменять задание при его редактировании пользователем. Свойство editedSelection будет содержать редактируемое значение, возвращаемое из контроллера деталей, как мы уже описывали.

Затем синтезируем get- и set-методы для свойства editedSelection.

```
.
.
.
@implementation BIDTaskListController
@synthesize tasks;
@synthesize editedSelection;
```

Немного изменим метод viewDidLoad, заменив старый тип NSArray новым типом NSMutableArray.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
.
.
.
~~self.tasks = [NSArray arrayWithObjects:~~
 self.tasks = [NSMutableArray arrayWithObjects:
.
.
.
```

В заключение мы собираемся реализовать свой set-метод для свойства editedSelection. Он заменит set-метод, который был неявно создан, объявлением @synthesize. Данный метод можно поместить в конец файла перед объявлением @end.

```
- (void)setEditedSelection:(NSDictionary *)dict {
    if (![dict isEqual:editedSelection]) {
        editedSelection = dict;
        NSIndexPath *indexPath = [dict objectForKey:@"indexPath"];
        id newValue = [dict objectForKey:@"object"];
       [tasks replaceObjectAtIndex:indexPath.row withObject:newValue];
       [self.tableView reloadRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
                                 withRowAnimation:UITableViewRowAnimationAutomatic];
    }
}
```

Этот метод выталкивает из массива заданий индекс и значение (редактируемое пользователем), которые будут возвращены в точку вызова. Затем он заталкивает в соответствующее место массива заданий новое значение и перезагружает соответствующую ячейку, чтобы в ней отображалось правильное значение.

Снова запустите приложение, перейдите к списку заданий, извлеките задание и отредактируйте его. Когда нажмете кнопку возврата, вы увидите, что отредактированное значение заменило старое значение в списке. Более того, поскольку соответствующая ячейка действительно перезагружается, тип ячейки можно изменить с `plainCell` на `attentionCell`, и наоборот, в зависимости от редактируемого значения. Попробуйте добавить слово URGENT к заданию, которое его не имело, или удалить его из названия задания и посмотрите, что получится.

Закончим работу гладким переходом

Теперь, изучив раскадровки, что вы о них думаете? Мы считаем, что это превосходный механизм для навигационных приложений. Мы собираемся использовать его в нашей книге еще несколько раз. Как указывалось в начале главы, работа с раскадровками на момент написания книги была возможна только в операционной системе iOS 5 и более поздних версиях. Это значит, что круг пользователей этого механизма ограничен теми, кто обновил систему или купил новое устройство. Со временем ситуация изменится, и все больше людей оценят мощь этой технологии.

Продвигаясь вперед, мы должны перейти к главе 11, в которой рассматриваются вопросы навигации, связанные с контроллерами представлений, специфичными для устройств iPad.